НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики та обчислювальної техніки

Обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____Сергій СТІРЕНКО

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Система перевірки лабораторних робіт студентів з програмування»

Виконав:
студент IV курсу, групи ІО-64
Федосов Андрій Олександрович _____

Керівник:
доктор технічних наук, професор
Новотарський Михайло Анатолійович _____

Консультант з нормоконтролю:
доктор технічних наук, професор
Сімоненко Валерій Павлович _____

Рецензент:
кандидат технічних наук, доцент
Орлова Марія Миколаївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Федосову Андрію Олександровичу

1. Тема роботи «Система перевірки лабораторних робіт студентів з програмування», керівник роботи Новотарський Михайло Анатолійович, доктор технічних наук, професор, затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи: технічна документація, теоретичні дані;

4. Зміст роботи: огляд існуючих методів рішення поставленої задачі, обґрунтування засобів розробки розроблюваної системи, опис функціональних можливостей та структури розроблюваної системи, демонстрація роботи розроблюваної системи;

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): принципова схема алгоритму перевірки лабораторних робіт студентів, структурна схема сервісу для перевірки лабораторних робіт студентів, функціональна схема класів системи для перевірки лабораторних робіт студентів.

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	д.т.н, проф. Сімоненко В. П.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми роботи	01.09.2019	
2	Вивчення та аналіз завдання	20.01.2020 – 09.02.2020	
3	Розробка загальної структури системи	10.02.2020 – 01.03.2020	
4	Програмна реалізація системи	02.03.2020 – 26.04.2020	
5	Тестування та виправлення помилок	27.04.2020 – 03.05.2020	
6	Оформлення пояснювальної записки	04.05.2020 – 25.05.2020	
7	Передзахист	26.05.2020	
8	Захист	15.06.2020	

Студент Андрій ФЕДОСОВ

Керівник Михайло НОВОТАРСЬКИЙ

Анотація

Дана дипломна робота присвячена розробці он-лайн сервісу для віддаленої перевірки лабораторних робіт з програмування студентів викладачем, а саме програмної частини роботи.

Даний сервіс допомагає автоматизувати процес прийняття лабораторної роботи. Завдання на роботу, програмний код та результат його виконання можна передивлятися прямо на сайті. Викладач має змогу перевірити роботу студента до заняття в аудиторії і вказати на її недоліки, помилки чи зарахувати її як виконану. Система дає змогу виконати завдання та перевірити роботу програми без завантаження додаткового програмного забезпечення. Таким чином, вчитель має більше часу на перевірку теоретичних знань студента.

Аннотация

Данная дипломная работа посвящена разработке онлайн сервиса для удаленной проверки лабораторных работ по программированию студентов преподавателем, а именно программной части работы.

Данный сервис помогает автоматизировать процесс принятия лабораторной работы. Задание на работу, программный код и результат его выполнения можно просматривать прямо на сайте. Преподаватель имеет возможность проверить работу студента до занятия в аудитории и указать на ее недостатки, ошибки или зачесть ее как выполненную. Система позволяет выполнить задание и проверить работу программы без установки дополнительного программного обеспечения. Таким образом, преподаватель имеет больше времени на проверку теоретических знаний студента.

Annotation

This diploma thesis is devoted to the development of an online service for remote verification laboratory works of students in programming by teacher, namely the program part of the work.

This service helps to automate the process of acceptance of laboratory work. Tasks, program code and the result of its execution can be viewed directly on the site. The teacher can check the student's work before class in the classroom and point out its shortcomings, mistakes or make it complete. The system allows you to complete tasks and test the program without downloading additional software. Thus, the teacher has more time to test the theoretical knowledge of the students.

Опис альбому
до дипломного проєкту
на тему: «Система перевірки лабораторних робіт студентів з програмування»

Київ – 2020 року

№ рядка	Формат	Позначення	Найменування	Кількість	Примітка
1			Завдання на дипломний проект	2	
2					
3	А4	*ІАЛЦ.467100.001 ОА*	Опис альбому	1	
4					
5	А4	*ІАЛЦ.467100.002 ТЗ*	Технічне завдання	3	
6					
7	А4	*ІАЛЦ.467100.003 ПЗ*	Пояснювальна записка	59	
8					
9	А4	*ІАЛЦ.467100.004 Д1*	Додаток А	1	
10					
11	А4	*ІАЛЦ.467100.005 Д2*	Додаток Б	1	
12					
13	А4	*ІАЛЦ.467100.006 Д3*	Додаток В	1	
14					
15	А4	*ІАЛЦ.467100.007 Д4*	Додаток Г	21	
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					

					ІАЛЦ.467100.001 ОА
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*	
Розроб.		*Федосов А. О.*			*Система перевірки лабораторних робіт студентів з програмування*
Перевір.		*Новотарський М. А.*			
Н. Контр.		*Сімоненко В. П.*			*Опис альбому*
Затверд.					

Літ.	*Аркуш*	*Аркушів*
	1	*1*

НТУУ «КПІ», ФІОТ ІО-64

Технічне завдання
до дипломного проєкту
на тему: «Система перевірки лабораторних робіт студентів з програмування»

Київ – 2020 року

Технічне завдання до дипломної роботи

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку системи для автоматизації перевірки лабораторних робіт студентів з «Програмування».

Область застосування: перевірка програмного коду в лабораторних роботах студентів, без використання додаткового програмного забезпечення.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на створення єдиної он-лайн системи для перевірки лабораторних робіт студентів з дисципліни «Програмування».

3. МЕТА РОЗРОБКИ

Метою даної розробки є створення єдиної он-лайн системи для автоматичної перевірки лабораторних робіт студентів з дисципліни «Програмування».

4. ДЖЕРЕЛА РОЗРОБКИ

Основними джерелами для розробки слугують публікації на дану тему в Інтернеті, документації існуючих програм з функціоналом для рішення цього питання, документації сервісів з он-лайн навчання.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваної системи

- Можливість вчителя задати завдання для роботи в системі
- Можливість вводити та запускати програмний код он-лайн
- Можливість обирати мову програмування згідно з дисципліною
- Можливість перегляду завдання на лабораторну роботу
- Вивід результатів компіляції та виконання програми
- Перегляд результатів роботи викладачем
- Можливість оцінювання та коментування програми викладачем
- Доступність для викладача та студента

5.2. Вимоги до програмного забезпечення

- Google Chrome, або інший аналогічний браузер

5.3. Вимоги до апаратного забезпечення

- Комп’ютер з підключення до мережі Інтернет

6. ЕТАПИ РОЗРОБКИ

Вивчення літератури інших матеріалів	20.01.2020
Складання та узгодження технічного завдання	03.02.2020
Аналіз функціоналу розроблюваної системи	10.02.2020
Аналіз програмного забезпечення	17.02.2020
Розробка модулів розроблюваної системи	02.03.2020
Тестування та виправлення помилок	27.04.2020
Оформлення документації дипломної роботи	04.05.2020

Пояснювальна записка
до дипломного проєкту
на тему: «Система перевірки лабораторних робіт студентів з програмування»

Київ – 2020 року

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

IDE	(Integrated development environment) Інтегроване середовище розробки
LMS	(Learning Management System) Система управління навчанням
HTML	(HyperText Markup Language) Мова розмітки гіпертексту
CSS	(Cascading Style Sheets) Каскадні таблиці стилів
JS	(JavaScript) скорочена назва мови програмування
URL	(Uniform Resource Locator) Єдиний вказівник на ресурс
EF Core	(Entity Framework Core) скорочена назва фреймворка
JWT	JSON Web Token
JSON	(JavaScript Object Notation) Запит об’єктів JavaScript
ORM	(Object relation mapping) Об’єктно-реляційна проекція
СКБД	Система керування базами даних

ВСТУП

Будь-який учбовий процес, так чи інакше пов’язаний з виконанням певних лабораторних робіт чи інших домашніх видів контролю студентом або учнем, та подальшою їх перевіркою викладачем, будь то лабораторна робота в університеті, або домашнє завдання на курсах.

Оскільки, сучасний світ, так чи інакше, зав'язаний на постійному використанні хмарних систем, та систем комунікації через Інтернет, не дивно було б припустити, що навчальний процес також можна було б перевести в он-лайн режим, хоча б частково. Розглядаючи це на прикладі перевірки лабораторних робіт в університеті, можна зрозуміти, що не всім студентам вистачає виділеного часу на занятті для захисту своєї роботи. Тим паче, що лабораторні роботи з програмування, які вимагають від студента розробки програми для вирішення певної задачі, захищаються у два етапи, а саме, перевірка викладачем самої програми, та перевірка теоретичних знань студента. Якщо ж викладачу не подобається виконання лабораторної роботи, то студент повинен її перероблювати, що подовжує час здачі роботи, що веде до недотримання термінів, а отже і знижує кінцевий бал студента. Також, часом, студент не може перебувати на занятті з поважних причин, або надзвичайних ситуацій при яких відвідування університету заборонено, натомість здача лабораторної роботи он-лайн може викликати певні незручності, такі як встановлення додаткового програмного забезпечення викладачем, яке в майбутньому йому не знадобиться.

Для вирішення даних проблем, доцільно щоб викладач мав змогу перевірити виконання програмного коду, та його коректність віддалено, використовуючи єдину зручну он-лайн систему, при цьому не використовуючи зайвого програмного забезпечення, що дає більше часу для живого спілкування зі студентом на занятті. На даний час, можна використовувати спеціальні он-лайн сервіси для запуску та перевірки коду програми, або сервіси для спільної розробки програмного забезпечення.

Існує багато сервісів з подібними можливостями, при цьому, майже, всі вони мають недоліки для впровадження їх в роботу у навчальних закладах.

Отже, постає питання у створенні спеціального он-лайн сервісу для перевірки коду програми лабораторної роботи викладачем. Даний сервіс має бути зручним, та доступним, як для студента так і для викладача, не потребувати додатково встановленого програмного забезпечення, або певних витрат, підтримувати всі необхідні мови програмування, а також можливість зв'язку викладача і студента. Також, важливим аспектом є надійність сервісу, та можливість його модернізації.

В даній роботі наведені приклади сервісів, які надають вищевказані функціональні можливості, при їх аналізі були відібрані основні функції та розроблена програма, яка може бути впроваджена у навчальний процес, для збільшення навчальних можливостей та розвитку навчального процесу в університеті.

РОЗДІЛ 1

ОГЛЯД ІСНУЮЧИХ СЕРВІСІВ ДЛЯ ПЕРЕВІРКИ ЛАБОРАТОРНИХ РОБІТ З ПРОГРАМУВАННЯ

Головною частиною лабораторних робіт з програмування є написання програмного коду. Зазвичай, щоб запустити або перевірити певний код програмісту необхідно встановлювати спеціальне програмне забезпечення, а саме спеціалізовані IDE (Integrated Development Environment) – системи, що можуть містити компілятор, ці системи використовують програмісти різного рівня для розробки програмного забезпечення.

Більшість IDE спеціалізовані під конкретну мову програмування, що створює необхідність у встановленні декількох IDE для роботи з різними мовами. Деякі IDE можуть підтримувати написання програми різними мовами, але загалом всі вони так чи інакше створюють сильне навантаження системи при запуску на виконання складного алгоритму або низки програм, до того ж часто IDE займає великий об’єм простору пам’яті на комп’ютері користувача. Для збільшення можливостей у програмуванні коду програміст також може завантажувати додаткові бібліотеки, що також потребують певних ресурсів комп’ютера.

Однією з головних цілей розробки є перенесення функціональних можливостей, які надає IDE в он-лайн систему, з можливістю зв’язку викладачів зі студентом. Загалом, такі можливості можуть надати два види сервісів, це он-лайн компілятори, та системи LMS.

1.1. Он-лайн компілятори

Он-лайн компілятор дає змогу, за необхідністю та без перешкод, запустити та швидко перевірити програмний код , при цьому немає потреби у запуску спеціалізованого програмного забезпечення [1], або встановленні додаткових бібліотек. Даний вид сервісу допомагає у ситуаціях при яких

треба негайно скомпілювати код, але при цьому необхідне програмне забезпечення відсутнє, при перевірці знань на співбесідах, тощо. Крім того більшість сервісів дозволяють обмінюватись кодом, що дає змогу співпрацювати в групових або приватних сесіях.

Он-лайн компілятори частіше за все є універсальними, тобто, можуть підтримувати багато мов програмування, але також існують більш спеціалізовані, що поглиблює можливості програміста в певних сферах програмування на певних мовах.

На рис. 1.1. можна побачити загальну структуру побудови он-лайн компіляторів. Зважаючи на сервіс, зовнішній вигляд компіляторів може різнитися, але загальний принцип завжди однаковий.

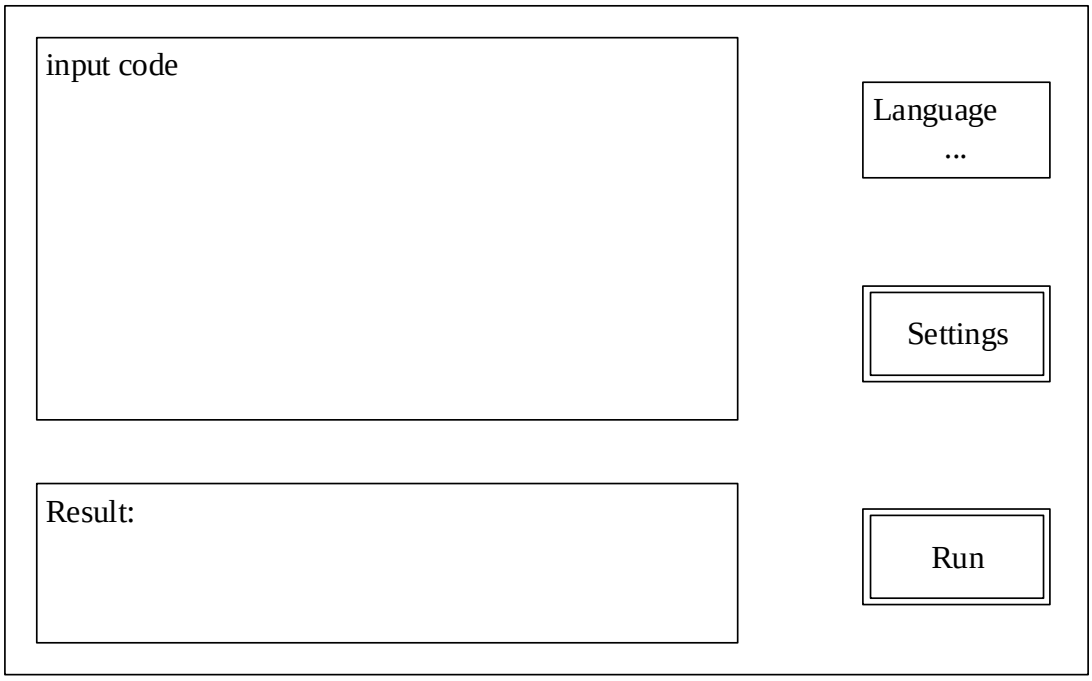


Рис. 1.1. Загальний вигляд інтерфейсу он-лайн компіляторів

Як показано на рис. 1.1 он-лайн компілятори містять поле для введення коду, в якому може підсвічуватися синтаксис мови, також, як додаток, деякі компілятори дають змогу завантажити файл з програмою прямо з пристрою. Більшість он-лайн компіляторів є універсальними, тому існує можливість для обрання мови програмування, і деякі додаткові налаштування. Нарешті є кнопка запуску програми, і поле для виводу результату. Загалом, поле

результату може містити результати по компіляції і результати програми, також може бути виведена інформація по вхідним даним, та ресурсам які використала система, наприклад кількість пам’яті, час виконання, або розмір файлу. Також сервіс може містити опцію збереження програми та результату, на пристрій, або в хмарне сховище.

Все що необхідно для роботи з таким сервісом, це Інтернет з’єднання. Часто он-лайн компілятори є безкоштовними, але інколи для відкриття спеціальних можливостей необхідна реєстрація, або оплата послуг при цьому для збереження або передачі програм між користувачами, завжди необхідна реєстрація. Загалом достатньо скопіювати, або набрати програмний код в спеціальне поле і натиснути одну кнопку, щоб отримати результат.

1.1.1. IdeOne

IdeOne – он-лайн компілятор [2], який користується високою популярністю в мережі. Даний сервіс є типовим представником універсальних он-лайн компіляторів, він підтримує понад шістдесят мов програмування в тому числі різні їх версії [1]. Система має вбудований налагоджував, що забезпечує можливість помічати помилки вже при написанні коду. Досить важливою особливістю системи є можливість вмикати, або вимикати підсвічування синтаксису, що може допомогти в навчанні, або тестуванні власних знань.

Головною причиною популярності даного сервісу є можливість зручної роботи групи людей над одним кодом. В залежності від режиму відображення коду, можна поділитися посиланням на цей самий код. Так існує можливість встановити три режими відображення коду, а саме:

- Public – будь-хто має відкритий доступ до коду.
- Secret – доступ відкритий тільки тим з ким автор поділився URL.
- Private – доступ до коду має тільки автор. [2]

Також, кожний, кому відкритий доступ, може залишити свій коментар, або зробити відгалуження, тобто оновити, модернізувати або просто продовжити роботу автора, за бажанням використовуючи іншу мову програмування, без зміни вихідного коду.

Для кожної програми можна задати вхідні параметри і відповідні їм перевірочні результати, що надає можливості як для навчання так і для зберігання інформації. За необхідністю, автор може додати вставки тексту, з певними замітками або описом програми.

Для авторизованого користувача, всі його коди можуть зберігатися на сайті, з відповідним рівнем доступу, стільки часу скільки необхідно, або завантажуватися з сайту. Система пошуку минулих робіт є доволі зручною та інтуїтивно зрозумілою, а спеціальні зразки програм, які доступні на перегляд кожному користувачу, допоможуть освоїти нові мови програмування, або закріпити існуючі знання.

Можливість перемикати відображення інтерфейсу у мобільний режим, надає змогу працювати навіть при відсутності комп'ютера, що дає незалежність від місця перебування користувача.

IdeOne містить певні обмеження, що забезпечують стабільну роботу системи, але при цьому зменшують можливості самого програміста. До таких обмежень відносяться: межа розміру вихідного коду в 64 kB, ліміт на час компіляції – 10 секунд, час виконання – 15 секунд, а також максимум використання 256 MB пам'яті [2].

1.1.2. JSFiddle

JSFiddle – це онлайн-редактор, який спеціалізується на створенні, здебільшого, різноманітних веб-розробок, а саме на їх інтерфейсах та інших не функціональних частинах. Даний сервіс є типовим представником спеціалізованих компіляторів для швидкої перевірки нових ідей або спільної роботи над одним проектом, він підтримує три мови програмування, а саме

JavaScript, HTML і CSS, при цьому надає більші можливості для роботи з ними і дозволяє виконати та поєднати їх роботу. Для більш поглибленої роботи у сфері веб-розробок, сервіс також підтримує додаткові фреймворки, такі як jQuery, React, Preact, React + JSX, Vue [1]. Також існує підтримка препроцесорів TypeScript, CoffeeScript, SCSS, які слугують для розширення можливостей в мові JavaScript та CSS, наприклад CoffeeScript дає можливість написати програмний код більш компактно ніж при використанні стандартного JavaScript. В можливостях сервісу також є підключення допоміжних інструментів для мови програмування CSS, таких як CSS Grid, Bootstrap та PostCSS [3].

Для зареєстрованих користувачів відкривається можливість збереження створених файлів, надання їм опису та коментарів. Користувач може зробити відгалуження від коду, тим самим змінюючи її, але при цьому не впливаючи на головну програму. Також, JSFiddle надає широкі можливості для зв'язку користувачів та колективної роботи з програмою. Існує можливість зробити URL посилання на ваші файли і поділитися ними зі своїми колегами, навіть незареєстрований користувач може відкрити посилання та подивитися його зміст. Якщо ж, необхідне живе спілкування між зареєстрованими користувачами, існує опція месенджера з можливістю написання сповіщення або розпочати голосову розмову.

Приємним додатком є надання широких можливостей для регулювання на власний смак візуальних налаштувань інтерфейсу робочого простору та початкових налаштувань мов програмування та їх фреймворків при запуску сервісу. За замовчуванням редактор поділено на чотири частини, три з яких відповідають мовам програмування, а четверта відображає результат [3]. Процес налаштування можна виконувати як у вкладці параметрів, так і безпосередньо на головній сторінці сайту. Сервіс також підтримує підсвічування синтаксису для більш зручної роботи.

Сервіс розповсюджується безкоштовно з деякими обмеженнями, але можна оформити підписку для розкриття всіх можливостей, таких як створення груп, в тому числі і приватних, вимкнення реклами, та безпосередньої можливості зв'язку з керівниками сервісу.

1.1.3. JDoodle

JDoodle не є звичайним он-лайн компілятором, а дає набагато більші можливості користувачу, а саме можливість відкривати он-лайн курси.

JDoodle – це універсальний інструмент, за допомогою якого можна швидко та легко перевірити свій програмний код різного рівня складності. Сервіс підтримує сімдесят дві мови програмування та різні їх версії, також є можливість роботі з двома базами даних, а саме MySQL та MongoDB [4]. На даний момент система підтримує додатковий репозиторій Maven для мови програмування Java, для інших мов використовуються стандартні бібліотеки, які постійно оновлюються. Зі всіх доступних в цих бібліотеках властивостей, не підтримуються тільки мережеві операції. Сервіс має зручний інтерфейс з підсвічуванням синтаксису мови, а також можливістю вибрати версію обраної мови. Інтерфейс містить поле вводу аргументів командного рядка, а також можливість переходити в інтерактивний режим.

При отриманні результату виконання програми, сервіс вказує час процесора, та пам'ять які використовує програма [4]. Після виконання відповідну програму можна зберегти, як на вашому аккаунті, так і завантажити на пристрій з якого ви зайшли на сайт. Сервіс, також, дозволяє редагувати раніше створені проекти, або завантажувати нові з файлу. Не збережені користувачем програми, залишаються доступними між сеансами, за допомогою можливості автономного локального збереження у браузері.

Для сумісної роботи над проектом існує можливість відкрити чат з іншими користувачами, або почати голосову розмову, при цьому доступ до

коду буде відкритий всім. Також збережений проект можна передавати між користувачами за допомогою URL посилання.

Основною відмінністю JDoodle від інших он-лайн компіляторів, э наявність спеціального сервісу JDoodle Guru. Це відгалуження являє собою можливість створення он-лайн курсів з програмування, так званих он-лайн інститутів або LMS. Цей сервіс дає змогу створювати та виконувати, як типові завдання у тестах, так і програмні додатки для рішення алгоритмів. Існують чотири типи ролей для користувача [5]:

- Власник – користувач, який відкрив курс. Він має повний доступ до всіх даних, змогу керувати всіма завданнями, та можливість змінювати параметри інших користувачів;
- Адміністратор – має майже всі ті можливості що й власник, але не може впливати на самого власника курсу.
- Викладач – може створювати групи і редагувати їх, після чого додавати студентів в ці групи. Викладач може розміщати навчальні матеріали, або завдання для групи, та редагувати їх. Завдання можуть бути як загальні так і окремі для кожного.
- Студент – користувач який може переглядати матеріали, та виконувати завдання.

Викладач може обрати один із стандартних типів завдання, таких як відповіді на питання, обрання правильної відповіді або відповідей з їх переліку, але сервіс дозволяє розробити завдання, відповіддю на яке буде програмний код студента з можливістю його запуску он-лайн.

JDoodle є безкоштовним, але з обмеженням по часу використання ресурсів сервісу. Дані обмеження зменшуються або знімаються при оплать відповідного тарифного плану.

1.1.4. Browxy

Browxy – це он-лайн компілятор, який підтримує сім мов програмування, в тому числі C, C++, C#, Java, Python та інші [6]. Browxy можна назвати найбільшим наближенням по функціональності до встановлюваного на комп'ютер IDE.

Головною властивістю даного сервісу є можливість створювати не просто один файл, а цілий зв'язаний проект, який може складатися з декількох репозиторіїв, які в свою чергу можуть включати кілька файлів з кодом, що розкриває можливості для побудови більш складних програм. Також є можливість для підтримки і завантаження бібліотек, що набагато розширює можливості користувача.

Browxy надає можливість зберігати та завантажувати проекти як на комп'ютер так і з нього, у вигляді архів [6]. Збережені файли можна опублікувати, або надати доступ окремим користувачам за допомогою URL адреси [6]. Також існує вбудований форум де можна знайти відповіді на деякі питання зв'язані з сервісом або програмуванням в цілому. Якщо необхідно, то є можливість створити власний чат для обговорення певної теми або проблеми.

Сервіс підтримує підсвічування синтаксису мови і авто-заповнювання, відображення інтерфейсу ж можна змінити на більш зручне для сприйнятя. Для зручного використання на будь-яких пристроях, існує можливість перемикати відображення сайту у мобільний режим.

Певно єдиним істотним обмеженням і недоліком даного сервісу є обмеження вільного місця для проектів, воно складає 20 MB, а один проект може займати максимум 10 MB, що суттєво зменшує можливості розробки [6].

1.2. Learning Management System (LMS)

LMS – це спеціальний сервіс для он-лайн навчання. Назва даного сервісу перекладається як «Система управління навчанням», та вказує на його основні принципи:

- Навчання(Learning) – можливість створити або долучитися до бажаних он-лайн курсів, з необхідною базою даних з певної теми. Тим самим розвиваючи себе, або навчаючи інших.
- Управління(Management) – полягає в управлінні курсами, завданнями, студентами і часом витраченим на навчання.
- Система(System) – надання свободи у можливості вибору місця проходження тих чи інших курсів. Автоматичний збір інформації про успішність студентів, перевірка тестових завдань та написання відгуків. [7]

Загалом дана система являє собою певний он-лайн університет, зі своїми власником, викладачами і студентами, базою даних та завданнями. Хоч навчання проводиться і он-лайн, але не обов'язково дистанційно, адже даний сервіс може бути додатком до навчання в університеті або іншому закладі і бути тільки допоміжним інструментом, для розповсюдження матеріалів та завдань.

В LSM сервісах існує певна система ролей. Звісно, один сервіс може відрізнятися від іншого, але кожний має включати принаймні три ролі:

- Автор – це та людина, яка створила курс, він може керувати навчальним процесом, додавати нові та редагувати старі матеріали, створювати групи та додавати в них користувачів.
- Адміністратор – це по суті викладач. Він може створювати та керувати групами, розробити власний план навчання, додавати нові матеріали, створювати завдання, додавати або видаляти користувачів з групи та змінювати ролі користувачів.

- Користувач – він же студент. Користувач може переглядати доступні матеріали, виконувати завдання, та долучатися до нових курсів. [7]

Загалом ці ролі можуть називатися по іншому, або об’єднуватися, так при невеликому курсі автор може бути єдиним вчителем, або вчитель на одному курсі може бути студентом на іншому в рамках одного інституту. В залежності від розмірів інституту та від тематики курсу ролей може бути більше.

В кожній системі LSM існує статистика успішності користувачів та часу витраченого на рішення того чи іншого завдання. Що стосується завдань, загалом вони поділені на звичайні тести, які система перевіряє автоматично та видає результати користувачу, та спеціальні відкриті питання, а для курсів, що включають в себе необхідність програмувати існує можливість підключити он-лайн компілятор [7] . Той, хто розробив завдання може настроювати терміни та час його виконання, з відповідною системою оцінювання, при чому результати тестування можна показувати, як одразу після проходження тесту, так і після проходження терміну здачі.

1.2.1. Moodle

Moodle – це типовий представник електронних систем навчання. Даний сервіс можна встановити, як в хмарній системі, так і на виділений сервер компанії або навчального закладу.

Існує кілька версій Moodle. Звичайна версія є безкоштовною і для її використання необхідна тільки реєстрація, при цьому велика частина можливостей стає недоступною, що зменшує варіації для можливого навчання користувачів, щоб повністю, або частково відкрити функціональні можливості сервісу, необхідно оформити підписку. Також, користувачі безкоштовної версії можуть користуватися тільки тими сервісами, які вбудовані в Moodle, використання сторонніх систем при побудові курсу дозволяється при наявності платної підписки [8]. Мобільна версія сайту

Moodle підтримує браузери типу Chrome та Safari, та надає можливість, як для проходження занять студентом, так і для виконання адміністрування з боку викладача. Існує також безкоштовний мобільний додаток, який дозволяє виконувати завдання в режимі оф-лайн, попередньо завантаживши їх на пристрій [8].

Основною силою Moodle є надання всіх можливостей для передачі матеріалу теми студенту. Гнучка система побудови лекції дозволяє розмістити текстові, графічні, аудіо та відео файли в зручному форматі. Після кожної лекції є можливість для створення тестів, при цьому в залежності від наявності підписки, можна використовувати, як вбудовані редактори, так і сторонні. Для курсів з програмування важливою особливістю є можливість підключення сторонніх он-лайн компіляторів, або інших сервісів.

Висновки до розділу 1

Проаналізувавши вищезазначені приклади сервісів, можна зробити висновок, що безкоштовні системи не надають повноцінної можливості для роботи викладача і студента, насамперед виконуючи тільки одну з поставлених задач, наприклад даючи змогу тільки запустити код при цьому зв’язок між вчителем і студентом буде проводитися з використанням сторонніх програм.

Ті ж сервіси, що існують для он-лайн навчання, є досить важкими для сприйняття або потребують грошових витрат з боку вчителя чи студента, при цьому надаючи надмірний надлишок функцій, що, звичайно, доповнює систему та надає нових можливостей, але не потребується наразі. Такі додаткові функції можуть впроваджуватися поступово, по мірі необхідності.

Таким чином, є потреба в створенні сервісу, який включав би основні можливості, які надають вищезазначені сервіси, при цьому не перевантажений нічим зайвим і доступний для кожного.

РОЗДІЛ 2

ЗАСОБИ РОЗРОБКИ СИСТЕМИ ПЕРЕВІРКИ ЛАБОРАТОРНИХ РОБІТ СТУДЕНТІВ З ПРОГРАМУВАННЯ

Першим питанням, яке виникло при розробці системи стало питання його реалізації, система може бути автономною у вигляді додатку на комп'ютері, використовуючи ресурси користувача, або мати ті ж серверні функції, що і сайт – запуск, перевірка, збереження та передача даних роботи студента, але працювати, як окремий додаток, такий собі он-лайн-компілятор встановлений на комп'ютер. Від цього шляху розвитку було вирішено відійти в сторону конкретного веб-сервісу, що має більше переваг над додатком ніж недоліків. Так однією з вимог було надання студентам та викладачам сервісу, що не потребує використання ресурсів їх комп'ютерів, при цьому він має надавати всі необхідні можливості для віддаленої роботи студента та викладача, як вказано в першому розділі, все це може надати веб-сервіс, при цьому всі обчислення будуть проводитися на виділеному сервері, що підтримується закладом. Також формат сайту надає можливість після проведення його модернізації, або його налагодження, надавати всім користувачам одну робочу версію, без необхідності її оновлення з боку студента або викладача, що не можна було б зробити використовуючи комп'ютерний додаток. Такий формат надає можливості для впровадження сервісу в спільну роботу з іншими сайтами навчального закладу, в їх підтримку та інтеграцію.

Структура сайту розділена на дві основні частини, так звані front end та back end, що відповідають за клієнтську та серверну частини відповідно.

Так клієнтська частина складає так званий інтерфейс користувача, обгортку програми, що допомагає йому координуватися в наданому сервісі та пов'язує його з серверною частиною, можна сказати, що це набір

керуючих пристроїв типу black box, що дає користувачу всю необхідну для нього інформацію і можливості, для виконання певних дій.

Серверна, або back end частина відповідає за всі дії та механізми сервісу, це загальна структура всього проекту і те, що по суті використовує будь-який користувач системи, його основа. Також всі дії , що виконуються для надання користувачу тих чи інших даних, наприклад зв’язок модулів, або бази даних, виконуються саме в цій частині.

Кажучи більш просто front end – те що бачить користувач, back end – те як все працює. Для сервісу є досить важливими кожна з цих частин, але якщо казати більш конкретно, то звичайно структура сайту, його back end, завжди стоїть у пріоритеті розробки. Інтерфейс користувача не завжди може бути занадто пророблений, якщо він дозволяє в повній мірі керувати функціональними характеристиками то цього буде достатньо, в той же час, як серверна частина повинна бути доведена до можливої досконалості.

Кожна із цих частин потребує власних засобів розробки, які б підходили для кожної вирішення поставлених завдань. До них відносяться, як мови програмування, загальні та спеціалізовані, так і додаткові технології, які розширюють можливості мов, на кшталт фреймворків та бібліотек, або надають абсолютно новий додатковий фунціонал для реалізації вирішення поставлених завдань і є абсолютно стороннім додатком впровадженим в роботу системи. Для розробки проекту я обрав засоби, які добре пов’язуються та працюють один з іншим, не останнім критерієм для вибору були також доступність цих засобів.

2.1 C#

Для створення системи була обрана мова програмування C#, загалом це об’єктно-орієнтована, строго типізована мова програмування високого рівня [9], що має велику кількість додаткових бібліотек та відгалужень, необхідних при побудові розроблюваної системи. Об’єктно-орієнтованість мови робить

необхідним створення опису кожної конструкції та взаємодії між ними, але при цьому дозволяє абстрагуватися від певної інформації, по суті використовуючи технологію чорного ящика, адже розробивши конструкцію один раз до неї можна не повертатися, допоки вона виконує свою функцію. Строга типізація дозволяє знаходити певні помилки ще на стадії написання програми, що по-перше захищає розробника від допущення цих помилок, а по-друге економить досить велику кількість часу, на пошук рішення цих помилок. Також дана мова є досить простою та зручною у використанні, вона містить досить велику кількість синтаксичного цукру, по суті вже готового рішення для поставленої задачі, що можна взяти, як повну альтернативу написання програми, або брати цю конструкцію за основу і якось модернізувати її, надбудовуючи над нею нові конструкції, розширюючи тим самим можливості свого коду. Істотним плюсом, який полегшує роботу з мовою C# є наявність великої кількості бібліотек та шаблонів, що розширюють можливості мови, та надають готові рішення, які в більшій мірі будуть більш оптимізованими ніж власноруч розроблена програма, тож не потрібно буде «заново винаходити велосипед», також існує велика кількість навчального матеріалу, де можна знайти відповідь на будь-яке питання стосовно цієї мови.

C# є повністю універсальною мовою, її можна застосовувати у більшості із IT сфер, та реалізовувати різні ідеї використовуючи тільки одну мову та різні її додатки та розширення, а можливість сумісної роботи та зв’язку з іншими технологіями дозволяє забезпечити необхідну функціональність для побудови веб-сервісу, для запуску розроблюваної системи.

Саме ці фактори і вплинули на обрання саме цієї мови програмування при розробці системи.

2.2. JS

Додатком до мови C# була обрана мова JavaScript, як одна з найрозвиненіших мов для створення веб-сайтів, що надає величезні можливості розробнику у створенні необхідних пристроїв для керування сторінкою та широким доступом до застосування додаткових об’єктів [10]. JavaScript є одночасно об’єктно-орієнтованою мовою і мовою яка підтримує прототипування, з динамічною типізацією. Загалом ця мова є деякою зборкою з різних мов, вона одночасно і схожа, і не схожа на інші. Дана мова проста в роботі, але не можна сказати, що і в сприйнятті також, все ж таки гарний код це не те чим вона може хизуватися. JavaScript містить велику кількість необхідних функцій, що надають об’єктам на веб-сторінці нових можливостей, загалом ця мова була створена, щоб покращувати роботу сайтів, робити їх більш інтерактивними, маніпулювати їх роботою та взаємодією [10]. Сильною перевагою цієї мови є її відносно проста робота в різних додатках з різними мовами, повна інтеграція з мовами HTML та CSS [10], що є основою будь якої веб-сторінки. Головною особливістю даної мови, що позитивно позначилась при виборі front-end мови, є наявність спеціальних фреймворків та бібліотек, що полегшує роботу як з самою мовою, так і з іншими технологіями, збільшуючи їх гнучкість та згладжуючи певні недоліки в синтаксисі.

Таким чином саме наявність спеціалізованих фреймворків та досить легка робота з іншими мовами в купі з доступністю у використанні, вплинули на вибір саме JS.

2.3. Vue JS

Раніше вже було сказано, що мови програмування обиралися з огляду на їх фреймворки, так одним з найпопулярніших JavaScript фреймворків для створення користувацького інтерфейсу є Vue JS, який базується чисто на

представницькому рівні [11] . Даний фреймворк комфортно використовувати при різних задачах, чи то одно-сторінковий додаток, чи високо адаптований інтерфейс користувача [11]. Так він використовує вже стандартні JS, HTML та CSS при створенні своїх компонентів, при цьому по суті підсилюючи ці технології, наприклад посилюючи роботу HTML оптимізовуючи її блоки. При цьому Vue може з легкістю працювати з іншими технологіями, не змінюючи своєї потужності. Його інтегрованість дозволяє використовувати даний фреймворк в будь-яких типах веб-сервісів при їх створенні, або впровадженні нових елементів у інфраструктуру вже створеного проекту без зміни працездатності останнього. Висока продуктивність Vue JS досягається за рахунок високої швидкодії, то ж його буде достатньо для проектів будь-якого розміру від одно-сторінкового до складного веб-сервісу, навіть якщо вони будуть далі розвиватися. Основна бібліотека Vue JS має невеликий розмір, до 20КБ, що не заважає зберігати гнучкість та повноту функціональності, а тільки допомагає забезпечити швидкість та продуктивність ставлячи даний фреймворк вище інших. При цьому існують додаткові бібліотеки, вільні для завантаження, які ще більше розвивають і без того гнучкі функціональні можливості.

Головною особливістю, на яку було звернуто увагу при виборі є низький поріг входження в роботу з Vue JS. Він досить легкий та простий при навчанні, з ним можна одразу приступати до роботи, не віддаючи багато часу на розбір синтаксису. При умові, що розробник має хоча б базові знання про мови JavaScript та HTML, він зможе працювати і з Vue JS. Фреймворк має велику кількість навчального матеріалу та базу користувачів, то ж відповідь на будь-яке питання стосовно його використання можна знайти в мережі, при цьому він містить документацію, яка є настільки детальною, що частіше за все, вистачає тільки її для вирішення певних проблем. Загалом новому користувачу достатньо тільки документації для навчання та використання Vue JS.

2.4. ASP.NET Core

Ще одним фреймворком для створення інтерфейсу веб-сторінки було обрано ASP.NET Core. Цей фреймворк є крос-платформним [12], що дозволило його використання не зважаючи на систему та не міняючи сферу розробки, а також надання широкого спектру можливостей для створення веб-сервісів різної складності, від легких веб-сайтів до складних порталів. Він має відкритий вихідний код, що вільно можна завантажити з файлів на GitHub, що спрощує розробку та доступ до зв'язування компонентів фреймворку, та відрізняється від інших спрощеною, але точною системою компіляції та тестування. Використання кодування моделі Razor Pages надає більшої продуктивності та полегшує створення інтерфейсів, роблячи кожну сторінку автономною [12]. Як вже було сказано, ASP.NET Core має велику функціональність побудований на компонентах, які майже не залежать один від одного, які через свою модульність можуть бути завантаженими окремо, тим самим даючи можливість користувачу самостійно обирати, як їх використовувати. Розробник може створити свій власний компонент зі своїми функціями та використовувати його, або змінюючи вже вбудовані у фреймворк компоненти по своїм побажанням. ASP.NET Core є досить складним, але наявність досить доброї документації та достатньої кількості інформації в мережі полегшують процес ознайомлення та роботи.

Хоча даний фреймворк не є достатньо легким в освоєнні, але його функціональність, можливість розширення та можливість інтеграції з іншими технологіями вплинули на вибір, адже кожний компонент системи повинен правильно працювати в купі з іншими.

2.5. PostgreSQL

При розробці сервісу який має оперувати певними даними, не дивно що виникає потреба в створенні бази даних. Однією з найкращих систем для

керування базами даних з відкритим вихідним кодом є PostgreSQL. Об'єктно-реляційна модель надає певні переваги даній системи над іншими, при цьому надаючи майже всі можливості, що й інші бази даних, тобто стаючи більш універсальною для роботи. Дана система надає широкі можливості у використанні типів даних, частіше за все різні бази використовують основні та деякі додаткові типи, PostgreSQL в свою чергу підтримує більшість існуючих типів даних та їх варіацій [13], крім того користувач може створювати власні типи для збільшення можливостей системи, а також використовувати масиви, які підтримуються системою. Серйозною особливістю, яка робить PostgreSQL кращою за інші є можливість створення користувачем своїх операцій, доменів, індексів, об'єктів та їх поведінки [13]. Ці особливості надають даній системі сильної гнучкості. Також PostgreSQL є, певно, найнадійнішою системою, адже при перевірці на помилки вона показала всього двадцять проблем на більше ніж пів мільйона рядків коду, що є серйозною перевагою над іншими. Щодо розмірів бази, дана система має мінімальні обмеження, так PostgreSQL не обмежена в розмірі бази даних, кількості записів та індексів, але при цьому має більше обмеження в кількості полів в одному записі в порівнянні з іншими системами.

Основними причинами у виборі цієї системи керування базами даних були її надійність, та доступність. Розширюваність та велика місткість системи звичайно надає більшої різноманітності у роботі з базами даних, але серйозного використання, наразі, ці особливості не набули, тим не менше всі вони будуть добре показувати себе при повноцінному використанні.

2.6. Docker

Розроблювана система основною функцією має запускати код на перевірку, тобто виводити результати його роботи та компіляції, з цієї причини сервер повинен мати функції компілятора. Запуск сторонньої програми на сервері може викликати непередбачувані наслідки, адже вона

може містити код, який впливає на операційну систему або її компоненти. Результатом виконання такого коду може бути як злам системи так і втрата певних даних з серверу. Тож постала необхідність у використанні технології, яка б забезпечила безпеку серверу при запуску стороннього коду. З огляду на цю проблему було вирішено скористатися технологією Docker.

Технологія Docker застосовує принципи віртуалізації, а саме контейнеризації, процесу розбиття ресурсів системи на кілька частин, так званих контейнерів, або зон [14]. Такі зони можуть використовуватися, як для запуску одного додатку, так і для виконання системи операцій, як в повноцінній операційній системі. В основі кожного контейнеру лежить образ, що по суті являє собою образ дистрибутива Linux [14], або інший створений на його основі, з додаванням певної функціональності, такий принцип дає змогу створення системи з файлами що необхідні тільки для виконання поставленої для контейнера задачі, без надмірного навантаження, тобто кожний контейнер запускає тільки необхідний додаток без завантаження додаткової операційної системи, таким чином роблячи Docker легким. Ці образи можуть бути створені власноруч, або завантажені з мережі, адже Docker є вільною програмою з відкритим кодом та великою користувацькою базою.

Розгортання контейнерів займає малу кількість часу та ресурсів системи, тим самим дозволяючи використовувати одночасно велику їх кількість без втрати продуктивності. Кількість виділених на один контейнер системних ресурсів можна корегувати та встановлювати самостійно, що по суті також є певною захисною дією з боку серверу, яка надає можливість обмежувати користувацькі програми з точки зору їх витратності і блокувати недопустимі додатки.

Як було сказано раніше, основною проблемою запуску коду на сервері є забезпечення безпеки основної системи. Контейнери Docker є абсолютно ізольованими в цьому плані, вони не мають доступу ні один до одного, ні до

системи-хоста. Таким чином кожний контейнер працює тільки зі своїм набором ресурсів, які надані йому системою [14], не впливаючи на роботу інших і навіть не маючи доступу до перегляду процесів, які проходять в інших контейнерах, що видаляє можливість навмисного або ненавмисного втручання в роботу системи, втрати даних, або зламу системи.

При закінченні роботи додатку в контейнері та наданні остаточних вихідних даних, цей контейнер видаляється не залишаючи після себе ніяких файлів, або іншої інформації. Такий підхід робить можливим безперервне використання системи, створення та видалення великої кількості контейнерів, без нагромадження непотрібних файлів, та забруднення простору системи.

Таким чином дана технологія дає змогу одночасно виконувати кілька програм на сервері без переймань, щодо його стабільності. Така технологія чудово пов’язується з поставленою задачею, адже контейнери мають виконувати роль компіляторів для різних мов програмування, які надаються навчальним закладом. При цьому вона містить вже створені образи для виконання цих дій, то ж все що необхідно це передати програму в контейнер, запустити її в ньому та отримати результат. Такої структури буде з надлишком достатньо для розроблюваного сервісу.

2.7. EF Core

Працювати з базою даних, безпосередньо, буває досить складно, то ж часто постає необхідність у використанні додаткових технологій, що спрощують дану роботу. Для спрощення роботи з базою даних проекту, було обрано використовувати технологію EF Core, яка дозволяє працювати з даними в базі за допомогою .NET об’єктів [15]. Дана технологія являє собою спеціальний інструмент, що називається ORM, який абстрагує роботу розробника від таблиць бази даних з їх індексами та ключами, та дозволяє роботу з об’єктами без залежності від їх типів, тим самим виводячи роботу

на вищій рівень абстракції, роблячи створення запитів більш зручною дією. Дана технологія не залежить від використовуваної системи керування базами даних, вона є універсальною, єдиним критерієм є наявність провайдеру потрібної системи. Таким чином, EF Core, через наявність універсального API, не потребує зміни коду програми для роботи з даними при переході від однієї системи до іншої. Як було сказано раніше, необхідною умовою роботи з системою керування базами даних є наявність її провайдера, технологія EF Core має кілька вбудованих провайдерів, серед яких є PostgreSQL провайдер [15], а саме ця система була обрана для роботи з базою даних у проекті, що, зрозуміло, позитивно вплинуло на подальшу роботу. При цьому, якщо виникне потреба в переході на іншу СКБД, EF Core підтримує деякі сторонні провайдери окрім тих, що вже встановлені в цій технології.

Керування запитами у EF Core відбувається через використання технології LINQ [15], що є по суті спрощеною мовою запитів на вибірку даних з бази. Використання цієї технології є особливістю EF Core, та головною рисою, що позитивно відрізняє її від інших. Кожен такий запит є універсальним для будь-якої СКБД, адже під час роботи з конкретною системою та виконанні певного запиту вираз з форми LINQ транслюється у спеціальну форму, яку розуміє ця СКБД. Дана технологія дозволяє створювати запити різної складності і не обмежує їх кількість, тим самим надаючи розробнику гнучку систему, що полегшує роботу з базою даних, без негативного впливу на останню.

EF Core була обрана на основі вже включених до проекту технологій, так ця технологія дає змогу керувати базою даних через .NET, а також містить вбудований провайдер для роботи з обраною для використання в проекті СКБД, що не потребує вивчення додаткових засобів. Дана технологія чудово співпрацює зі вже згаданими додатками, що полегшує розробку, та економить час.

2.8. VueX

При побудові складних систем на основі Vue JS, з часом, починають траплятися певні проблеми в управлінні станами, тобто даними якими керує програма. Щоб не засмічувати та не плутати власний код, було вирішено використати додаткову бібліотеку VueX. Ця бібліотека була спеціально створена для керування станами у проектах написаних за допомогою Vue JS [16]. VueX працює за системою сховища, яке містить в собі певний набір компонентів. Це сховище є централізованим та реактивним, що надає високої швидкості оновлення станів, майже миттєвої [16]. Стан сховища не може бути змінений компонентами через структуру сховища, таким чином зміна стану може відбуватися тільки так, як воно заплановано, що надає безпечності через запобігання руйнуванню станів, при отриманні запиту іншими компонентами. Зміна стану сховища відбувається через використання так званих мутацій – оновлень стану [16]. Така система дає можливість компонентам взаємодіяти один з одним, або бути видаленим без впливу на загальний стан.

Таким чином дана бібліотека була використана в додаток до фреймворку Vue JS для покращення структури збереження та зміни даних.

2.9. JWT

Для роботи в розроблюваній системі користувачу необхідно буде в ній авторизуватися, що призводить до необхідності в передачі конфіденційних даних користувача, тому було вирішено при побудові системи авторизації використовувати токени, даний формат є одним із найбезпечніших способів передачі інформації, а використання JSON технології робить його надійним. JWT об’єкт є самодостатнім через його структуру, яка складається з трьох частин, які називаються заголовок, навантаження або інформація та підпис [17], у вигляді JWT об’єкта вони вказуються через крапку у форматі JSON

[17]. Якшо розглядати ці частини окремо, то можна сказати так, заголовок надає інформацію про алгоритм хешування даних, навантаження містить інформацію та дані, яку необхідно передати за допомогою токена між двома сторонами, підпис створюється на основі кодування заголовку та навантаження, а також на використанні ключа, тобто пароля користувача. При першій авторизації система генерує для користувача JWT на основі його даних та запам’ятовує його, після цього при наступних запитах система перевіряє наявність відповідного заголовку та навантаження у базі, при їх наявності виконується створення підпису на основі наданого користувачем ключа, а потім цей підпис порівнюється з існуючим [17].

Дана технологія є досить простою, але дієвою, вона дає можливість безпечної авторизації, що безумовно є необхідною умовою в роботі розроблюваної системи.

Висновки до розділу 2

Для розробки системи для перевірки лабораторних робіт студентів з програмування були використані технології описані в даному розділі. Кожна технологія виконує свої певні функції, які вкупі створюють повністю робочу систему.

Основними критеріями у виборі розглянутого програмного забезпечення були легкість в використанні – всі технології мають розгорнуту документацію та поширену інформаційну оболонку, доступність – всі ці технології можна вільно використовувати та завантажувати ліцензійні версії, можливість спільного їх використання – вони мають високий рівень інтеграції та не потребують встановлення програм посередників. Тим самим при розробці вдалося уникнути серйозних складнощів та проблем.

Використання цих технологій дає можливість подальшому розвитку системи, без втручання у вже працюючі механізми, при цьому її буде досить легко підтримувати та модернізовевати.

РОЗДІЛ 3

ОПИС ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ ТА СТРУКТУРИ РОЗРОБЛЮВАНОЇ СИСТЕМИ

Спираючись на аналіз існуючих сервісів для перевірки програмного коду та систем дистанційного навчання, було виділено основні можливості які надає розроблювана система. Дана функціональність розділена між двома видами користувачів, викладачем та студентом, як вказано на рис. 3.1.

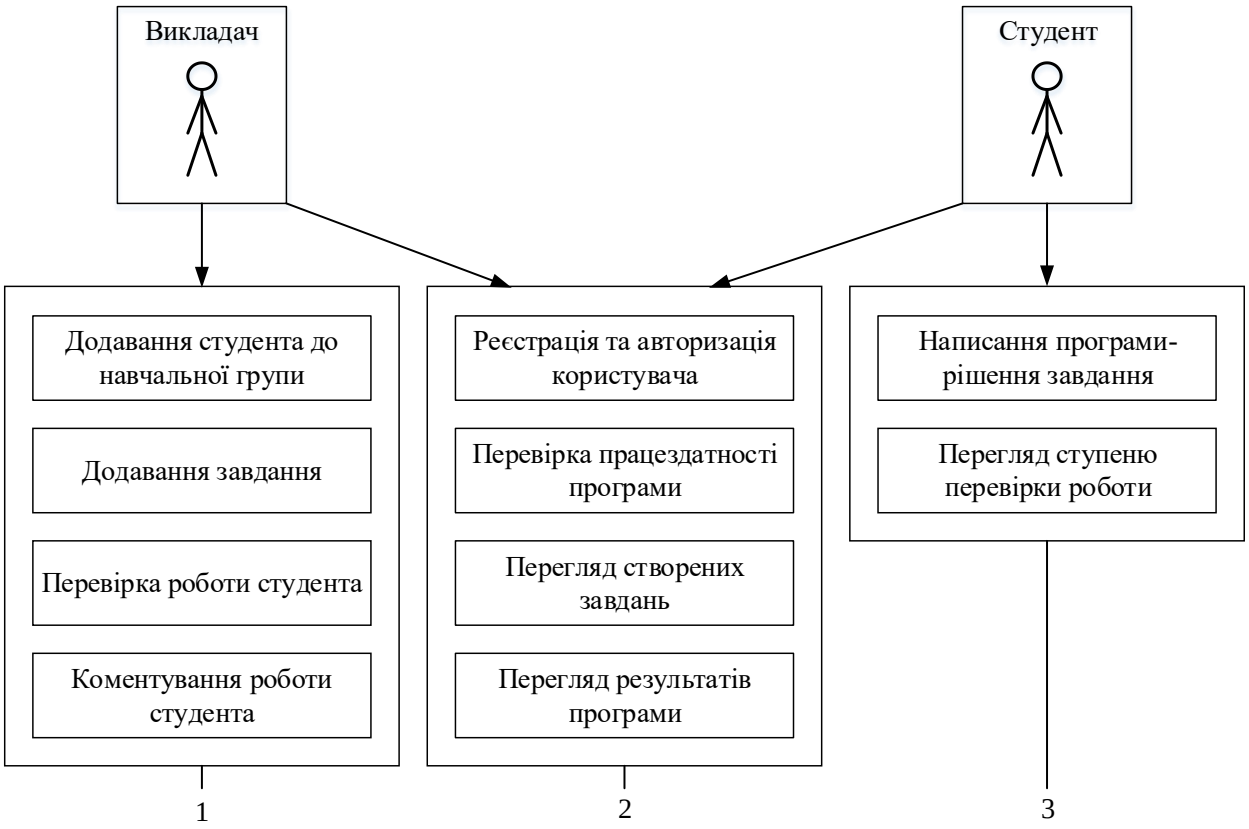


Рис. 3.1. Схема розподілення функціональних можливостей системи між користувачами:

1 – функції та дії, які може виконувати викладач;

2 – функції та дії, які може виконувати будь-який користувач;

3 – функції та дії, які може виконувати студент;

3.1. Функції та дії спільні для всіх користувачів

3.1.1. Реєстрація та авторизація користувача

Для початку роботи в системі, користувач має пройти реєстрацію, яка також необхідна для звітності при виконанні роботи. Під час реєстрації користувач може обрати одну з двох ролей, а саме викладач або студент, що буде впливати на його можливості в даній системі. Для подальшої ідентифікації особи, користувач, також, при реєстрації повинен ввести прізвище та ім’я латинськими літерами, а для подальшої повторної авторизації – логін та пароль. При наступному входженні в систему користувач виконує авторизацію використовуючи раніше введені данні.

3.1.2. Перевірка працездатності програми

Основною властивістю розроблюваної системи є можливість в он-лайн режимі запускати програмний код. Так, вбудований в систему компілятор дає змогу записати або скопіювати програмний код в спеціально відведене поле, та запустити програму на одній з доступних мов програмування. При компіляції проводиться перевірка на синтаксичні та семантичні помилки програми, та виводиться результат компіляції.

3.1.3. Перегляд створених завдань

Як студент так і викладач можуть переглядати завдання на лабораторну роботу. Перш за все студент повинен бачити своє завдання за варіантом і писати програму відносно нього, викладач в свою чергу може перевірити правильність вибору студентом варіанту та відповідність роботи програми до змісту варіанта завдання.

3.1.4. Перегляд результатів програми

При запуску програми на виконання, користувач бачить результати її роботи. Результат роботи програми можна переглянути, як в спеціальній вкладці чернетки, так і при написанні студентом відповіді на завдання. Також при отриманні роботи студента, викладач бачить не тільки програмний код, але й, одразу, результати компіляції і виконання програми, що дозволяє, при необхідності, не запускати, для перевірки, програму повторно.

3.2. Функції та дії викладача

3.2.1. Додавання студентів до навчальної групи

Для того щоб студент міг бачити завдання, та прикріплювати свої відповіді у вигляді програмного коду, викладач повинен додати його до своєї бази даних. Для додаванні студента, викладач використовує пошук по базі зареєстрованих користувачів на основі ім’я та прізвища студента.

3.2.2. Додавання завдання

Викладач може створювати завдання для своєї групи, а саме в текстовому виді описати загальне завдання для всіх, або розписати окремі варіанти, та принцип їх визначення.

3.2.3. Перевірка роботи студента

Викладач може переглядати всі результати виконаних студентами робіт, які вони додають до певного завдання. Він бачить програмний код, його автора, та результат виконання цього коду, також, при необхідності, можна ще раз запустити програму на виконання, перевіривши її результат з іншими вхідними даними. На підставі отриманих даних, викладач може оцінити роботу, як прийняту, або не прийняту натиснувши відповідну кнопку.

3.2.4. Коментування роботи студента

При оцінюванні роботи студента, викладач має змогу додати коментар, в якому може вказати недоліки, або помилки в програмі, або в її рішенні. Також, коментар дозволяє донести додаткову інформацію стосовно даної, або майбутньої роботи, наприклад оцінку, або наступний варіант.

3.3. Функції та дії студента

3.3.1. Написання програми-рішення завдання

При отриманні завдання, студент має можливість розробити програму для його рішення в он-лайн режимі. Після перевірки своєї програми, студент може додати її до списку результатів для певного завдання.

3.3.2. Перегляд ступеню перевірки роботи

Студент має змогу передивитися стан зарахування своєї роботи, а саме він бачить, робота перевірялася, чи ні, якщо перевірялася то зарахована вона чи не зарахована, та коментар викладача стосовно роботи, якщо він є.

3.4. Опис структури серверної частини системи

Система, що була розроблена в даній роботі складається з back-end частини, тобто внутрішньої структури (Додаток Г); front-end частини – візуального вигляду сервісу, та бази даних. Далі розглядається внутрішня структура системи, лістинг якої наведено в Додатку Г.

Структура системи розділена на дві частини, перша містить в собі логістику сервісу, основні класи програми та зв’язки між ними, ця частина включена в директорію WebApplicationBLL. В цю директорію входить декілька структур, а саме: класи сервісів (Services), інтерфейси (Interfaces) класів, валідатори (Validators) та файли DTOs.

1. Клас CodeService відповідає за запуск коду на сервері. Даний клас приймає на вхід код програми та мову програмування, на їх основі створюється файл з кодом (метод CreateCommandFile), після чого створюється файл docker-a (метод CreateDockerFile) та образ (метод RunDockerImageBuildScript), далі створюється контейнер та запускається програма збережена у файл (метод RunDockerImage), в результаті після завершення роботи програми контейнер видаляється (метод RemoveDockerImage), а результат програми видається як результат класу.

2. Клас TaskResultService відповідає за створення, перегляд та оцінювання результатів робіт. Даний клас має чотири методи: Create, GetTeacherTaskResults, GetStudentTaskResults та SetTaskResult.

- Метод Create приймає параметром зміну типу TaskResultCreateDto, яка містить всю інформацію про результат завдання та додає ці дані до сховища результатів.
- Метод GetTeacherTaskResults приймає параметрами ідентифікатор завдання (taskId) та викладача (teacherId), відповідно до цих параметрів із сховища надаються дані по всім результатам, які були додані до відповідного завдання.
- Метод GetStudentTaskResults працює так само як і метод GetTeacherTaskResults, але замість ідентифікатора викладача приймає ідентифікатор студента (studentId) і відповідно надає тільки ті результати, які відносяться до даного студента.
- Метод SetTaskResult відповідає за оцінювання роботи студента, він приймає параметрами ідентифікатор результату (taskResultId), викладача (teacherId), тип оцінки (type) та коментар (comment) залишений викладачем. На основі ідентифікаторів, зі сховища береться відповідний результат і до його даних додається оцінка та коментар, після чого він знову зберігається в сховище.

3. Клас TaskService містить три методи: Create, GetUserTasks та GetTeacherTasks і відповідає за створення та перегляд завдань.

- Метод Create приймає параметром зміну типу TaskCreateDto, ця зміна містить опис самого завдання (Description) та ідентифікатор користувача (UserId), на основі цієї інформації в сховище завдань додається нове завдання.
- Метод GetUserTasks по прийнятому ідентифікатору студента (studentId), знаходить всі завдання в сховищі до перегляду яких даний студент має доступ, зчитує їх дані та надає список зі всіх знайдених завдань.
- Метод GetTeacherTasks працює так само як метод GetUserTasks, але для викладача за його ідентифікатором (teacherId) та надає список із завдань, які створив він сам.

4. Клас UserService відповідає за дії які відбуваються безпосередньо з інформацією користувача, його авторизацію, реєстрацію та пошук користувача в базі даних.

5. Класи валідації (Validators) існують для перевірки наявності та допустимості значень, які були введені при заповненні певних полів форм.

6. Класи DTO являються нічим іншим, як описами об’єктів, які використовуються в системі.

Друга частина структури містить контролери для зв’язку back-end та front-end частин, та міститься в директорії WebApplication. В даній директорії містяться відповідно класи контролерів (Controllers), файли моделей (Models) та клас AutoMapperProfile.

1. Наявні три класи контролерів CodeController, TaskController та UsersController. При створенні певного HTTP запиту обирається відповідний йому контролер, виконується прив’язування моделі, та відображення даних

запиту у вигляді параметрів методу, після чого визивається відповідний метод і повертається його результат.

2. Класи Models являються описами об’єктів, які складаються на основі даних, наданих з сервісу користувачем.

3. Клас AutoMapperProfile зв’язує об’єкти описані в Models з об’єктами описаними в DTOs попередньої директорії.

Висновки до розділу 3

При аналізі існуючих розробок та сервісів, що мають можливості для перевірки лабораторних робіт з програмування, була розроблена функціональна характеристика для розроблюваної системи. Дана характеристика включає всі необхідні можливості для перевірки програмного коду лабораторних робіт студентів викладачем, без необхідності встановлення додаткового програмного забезпечення, або інших витрат. Усі функції направлені тільки на можливість віддаленої роботи студентів і викладача, без додавання зайвих функції, які можна спостерігати у інших сервісах.

Таким чином розроблювана система має достатню для рішення поставленої задачі функціональність. При цьому, при необхідності, даний можливості можна буде розширювати або змінювати.

РОЗДІЛ 4

ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ ДЛЯ ПЕРЕВІРКИ ЛАБОРАТОРНИХ РОБІТ СТУДЕНТІВ З ПРОГРАМУВАННЯ

Завданням даного дипломного проекту було створення системи, яка б допомагала викладачу перевіряти лабораторні роботи студентів з програмування, а саме перевіряти працездатність їх програм. На основі існуючих розробок, які вказані в першому розділі були розроблені функціональні характеристики системи, які вказані в третьому розділі та обрано програмне забезпечення, яке дозволило втілити цей їх.

В даному розділі продемонстровано роботу готової системи, яку можна впроваджувати в роботу в навчальних закладах. Поетапно буде показана робота всієї системи зі сторони викладача та студента, починаючи від реєстрації користувача і закінчуючи перевіркою роботи студента.

Перше що бачить користувач при вході в систему це форму для авторизації, де необхідно заповнити поля логіну та паролю (рис.4.1).

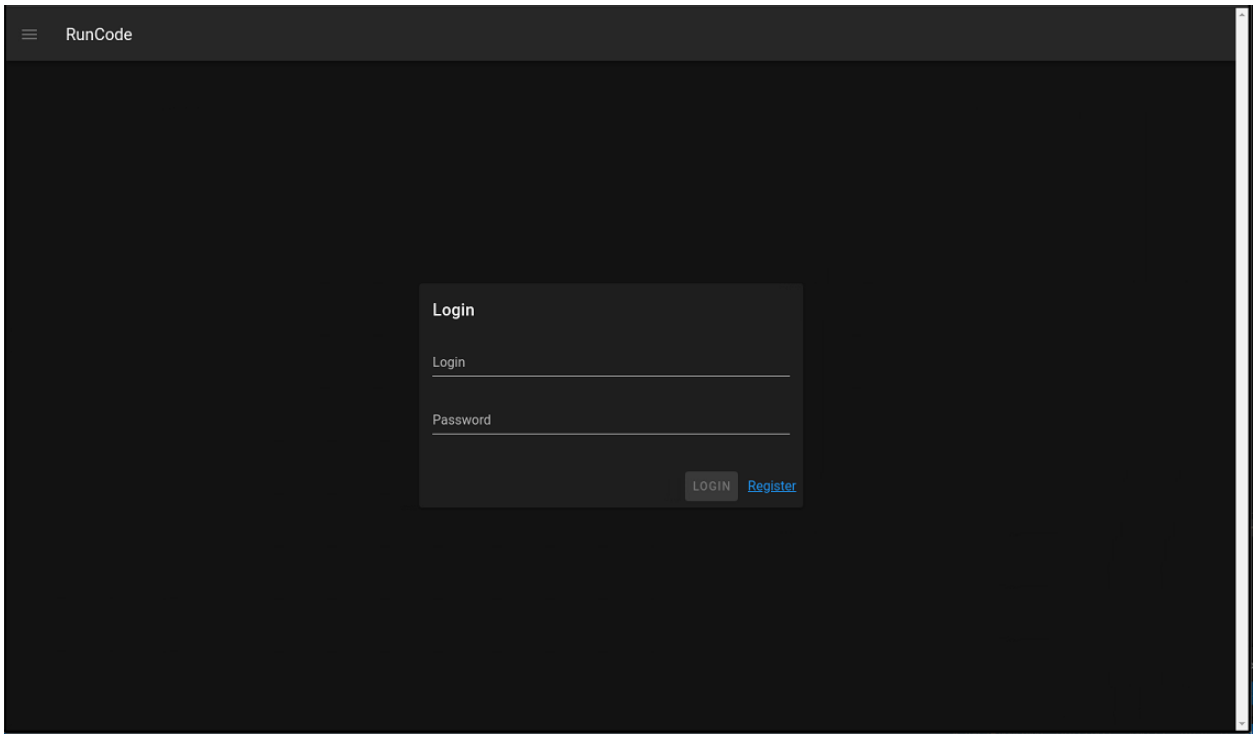


Рис. 4.1. Форма для авторизації користувача

Якщо ж користувач ще не зареєстрований, то він може це зробити натиснувши на посилання «Register» в правому нижньому куті форми. Натиснувши на посилання користувач переходить до форми для реєстрації (рис.4.2), яка містить чотири поля для заповнення та одне поле вибору. Поля для вводу є стандартними для будь якої реєстрації, це ім’я (FirstName), прізвище (LastName), логін (UserName) та пароль (Password), що слугують для ідентифікації користувача в системі, останнє поле може містити на вибір одне з двох значень, ролей: «Студент» («Student») або «Викладач» («Teacher»), від значення, яке обрано в цьому полі, залежать можливості які будуть надані користувачу для роботи в системі. При цьому значення які вводяться в поля мають містити тільки латинські літери, а довжина паролю має бути не меншою від шести символів.

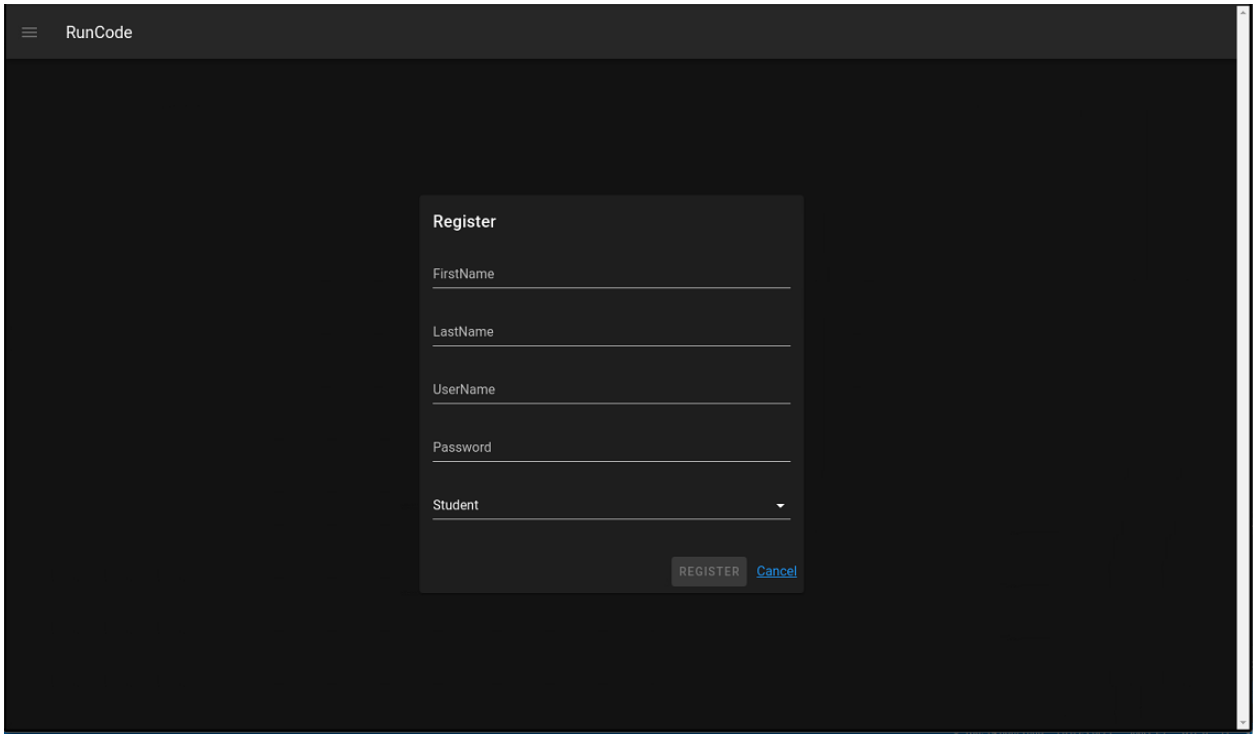


Рис. 4.2. Форма для реєстрації користувача

На рис.4.3. показано приклад заповнення форми реєстрації та обрано роль вчителя. Якщо всі поля заповнені успішно, тоді при натисканні на кнопку "REGISTER" в правому нижньому куті форми, вас повідомлять про

успішну реєстрацію та ви повернетеся на форму з авторизацією, де можна буде ввести свої дані (рис.4.4).

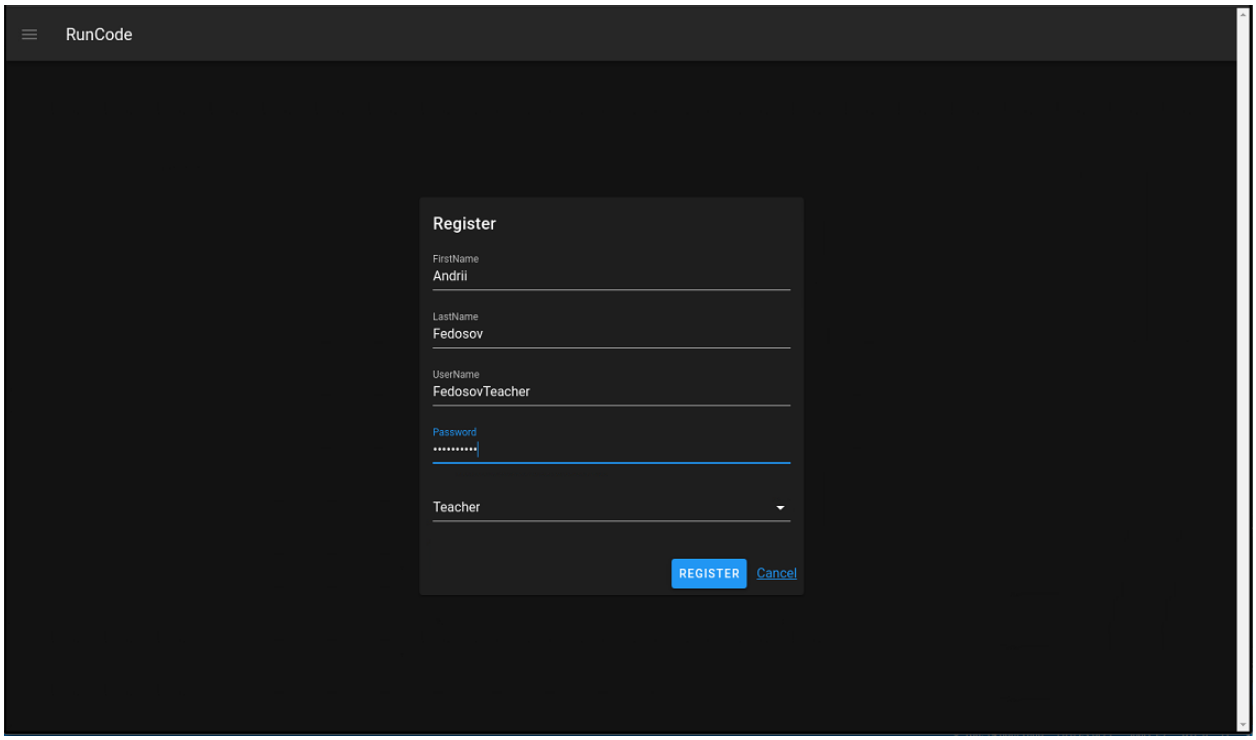


Рис. 4.3. Приклад заповненої форми для реєстрації

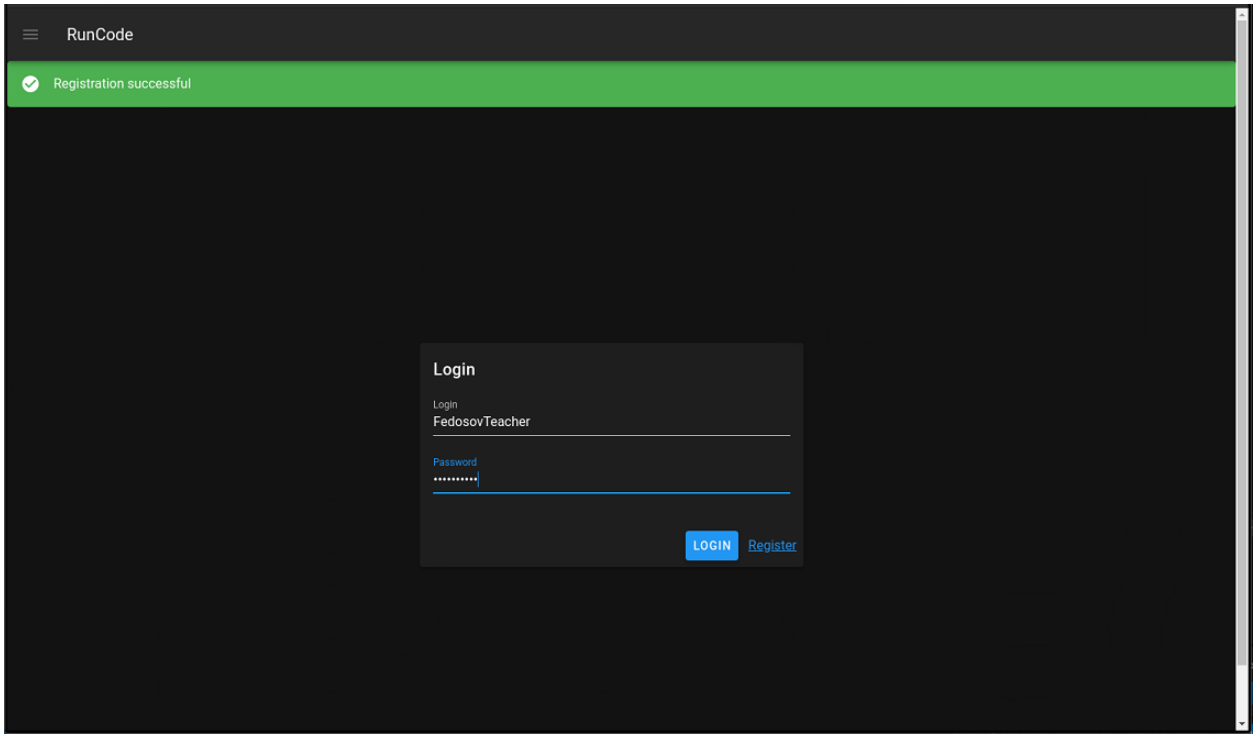


Рис. 4.4. Приклад заповненої форми для авторизації

4.1. Демонстрація роботи викладача в системі

Після успішної авторизації користувач бачить сторінку головного меню, для викладача вона виглядає так, як показано на рис.4.5.

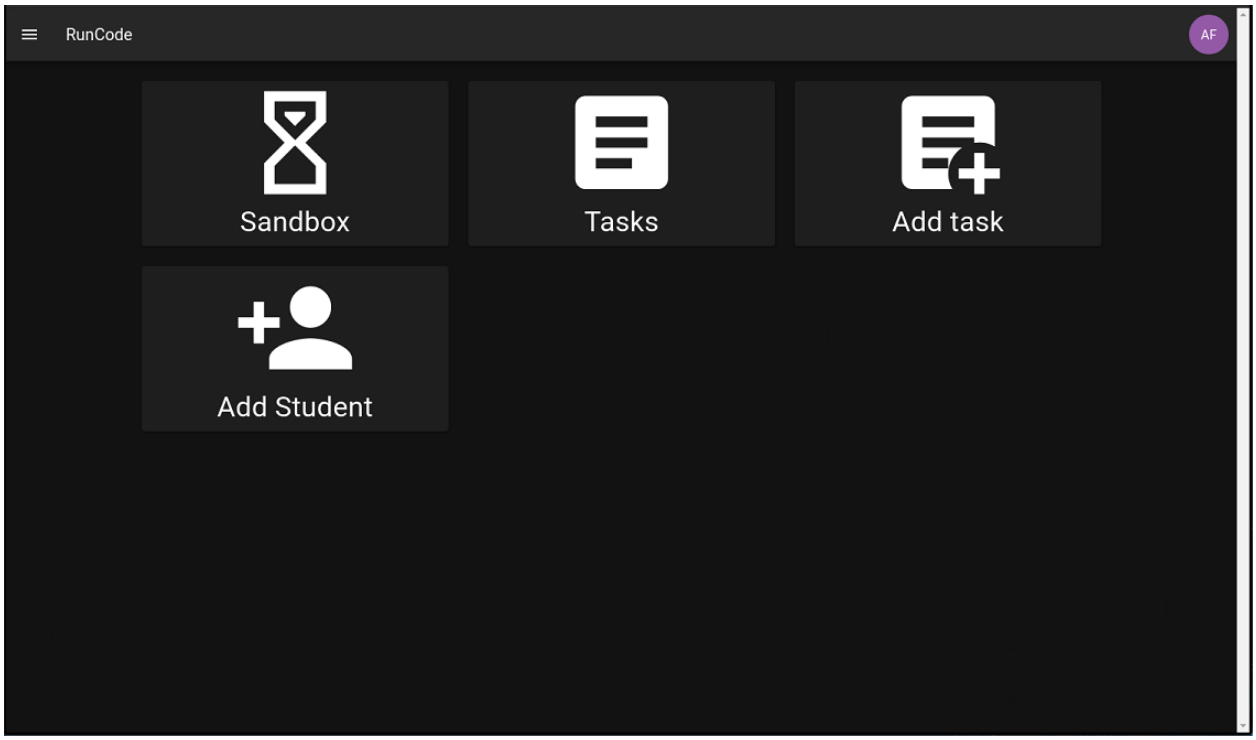


Рис. 4.5. Головна сторінка користувача «Викладач»

Головна сторінка викладача містить чотири вкладки (рис.4.5), кожна з них відповідає за певну сторінку на якій виконуються певні дії в навчальному процесі. Так, натиснувши на вкладку «Sandbox» відкриється сторінка де можна запускати та перевіряти роботу коду програми, вкладка «Tasks» веде на сторінку де викладач може переглядати завдання та відповіді, які надіслали студенти на ці завдання, після чого оцінювати їх, на сторінці, що відкривається натиском на вкладку «Add task», викладач додає нові завдання для студентів, ці завдання бачать тільки ті студенти, кого викладач додав до робочої групи (своєї бази студентів), натиснувши на вкладку «Add Student» викладач перейде на сторінку, через яку зможе додати студента у свою базу, щоб мати можливість з ним працювати, та перевіряти його роботи.

Користувач може розгорнути бокове меню (рис.4.6), натиснувши на значок в лівому верхньому кутку вікна, це меню доступне з будь-якої сторінки сервісу і виконує навігаційну функцію, оскільки воно містить посилання на всі інші сторінки дублюючи тим самим головну сторінку користувача. Також внизу бокового меню є кнопка «LOGOUT», натиснувши на яку, користувач може змінити обліковий запис.

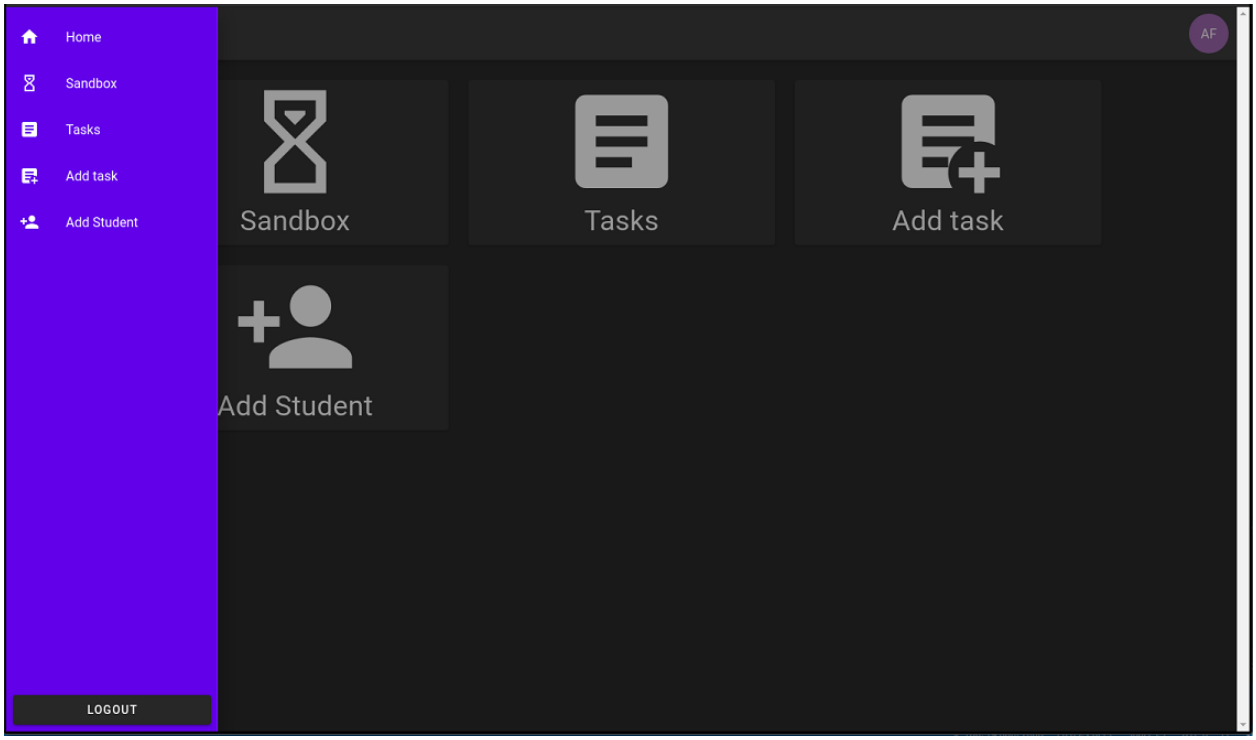


Рис. 4.6. Вигляд бокового навігаційного меню системи

При натисканні на вкладку «Sendbox» користувач переходить на сторінку перевірки програмного коду (рис.4.7), яка являє собою стандартний он-лайн компілятор, також за на цю сторінку можна перейти за допомогою бокового меню (рис.4.6). Дана сторінка в верхній частині містить поле вибору мови програмування, за замовчуванням обрана мова Python, але також можна обрати мову C# або Java, далі йде поле в якому записується код програми. Як вже було сказано, дана програма підтримує три мови програмування, але вона не є повноцінним компілятором, це зумовлено тим, що програмувати з нуля на даній сторінці не рекомендується хоча й можна, адже при закритті сторінки, або при її перезапуску код не збережеться, тому

доцільно використовувати її тільки для перевірки працездатності коду, наприклад, як поведе себе програма при використанні інших даних, або перевірити, чи не є програма пустишкою.

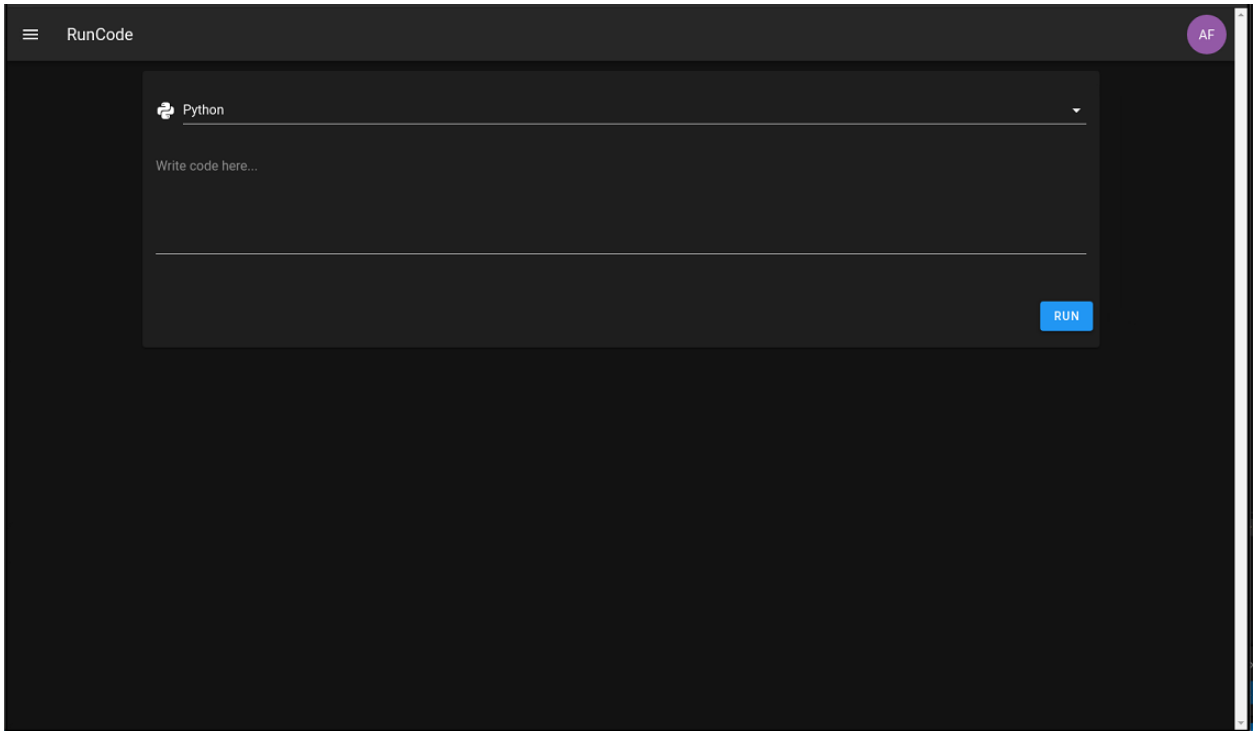


Рис. 4.7. Вигляд сторінки для запуску та перевірки програмного коду

Після вибору необхідної мови, користувач може записати код або скопіювати його в відведене під нього поле та перевірити результат його виконання натиснувши на кнопку «RUN». Після цього система почне обчислювати дану програму, а після завершення виведе результати обчислень та компіляції програми у поле «Result» (рис.4.8), яке автоматично додається на сторінку. При виникненні помилки компілятор вкаже на її наявність, номер рядка з помилкою, місце в рядку і її пояснення та виведе цю інформацію в полі «Result» відповідним сповіщенням (рис.4.9).

Компілятор підтримує основні стандартні бібліотеки кожної з доступних мов, тобто його цілком достатньо для перевірки програм, які є не надто складними та можуть бути запущеними з одного файлу без використання додаткових засобів на стаціонарному комп'ютері користувача.

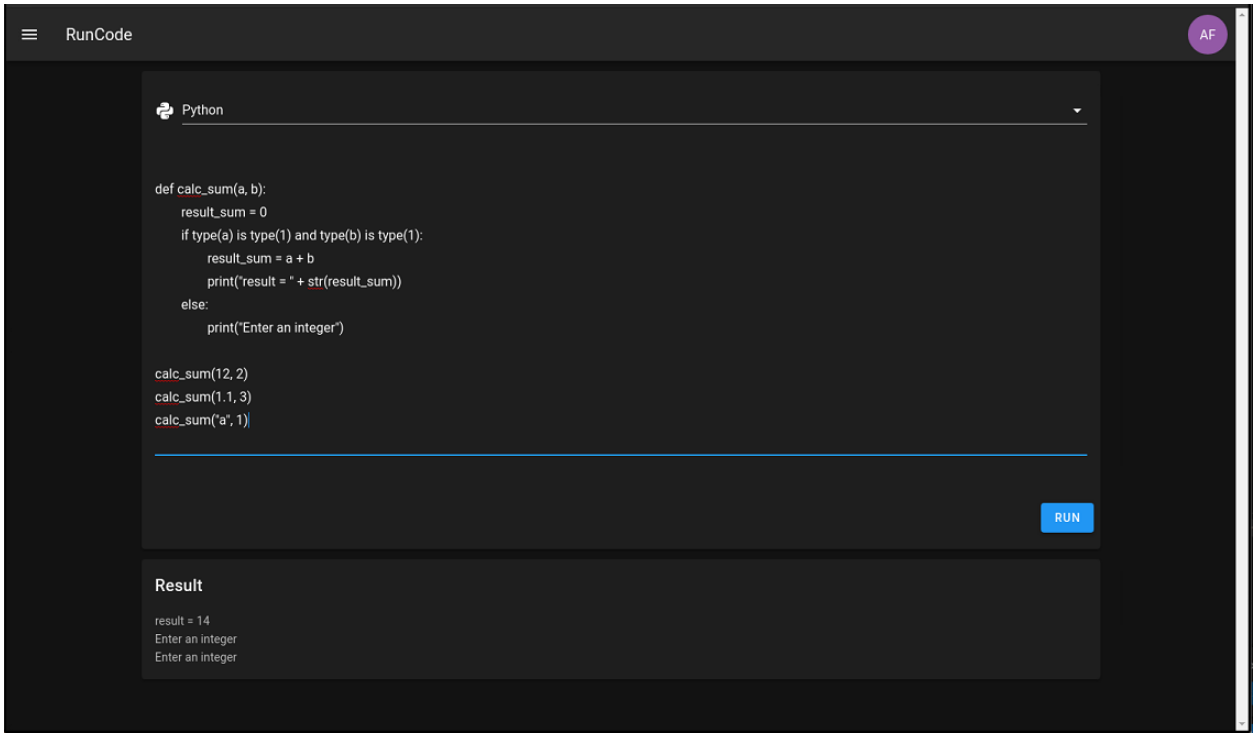


Рис. 4.8. Приклад роботи компілятора на сторінці для запуску та перевірки програмного коду

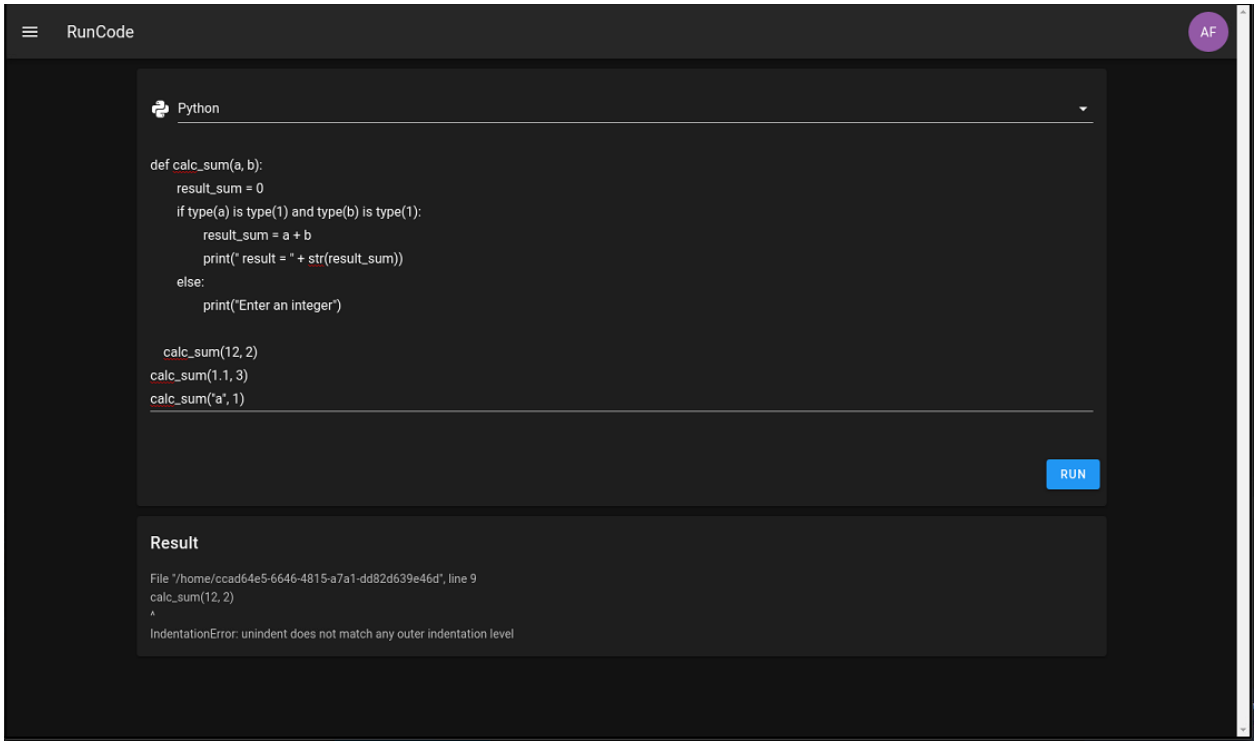


Рис. 4.9. Приклад виводу результату роботи програми з помилкою в коді

Натиснувши на вкладку «Add task» в головному або боковому меню вчитель перейде на сторінку, де він може додати завдання для своїх студентів (рис.4.10). Сторінка має поле для вводу в текстовому форматі завдання «Description», при заповнені якого підсвічується синім кольором кнопка «ADD» (рис.4.11) натиснувши на яку, завдання додасться в базу даних до інших завдань, а в якості підтвердження внизу сторінки з'явиться сповіщення про вдале збереження завдання (рис.4.12).

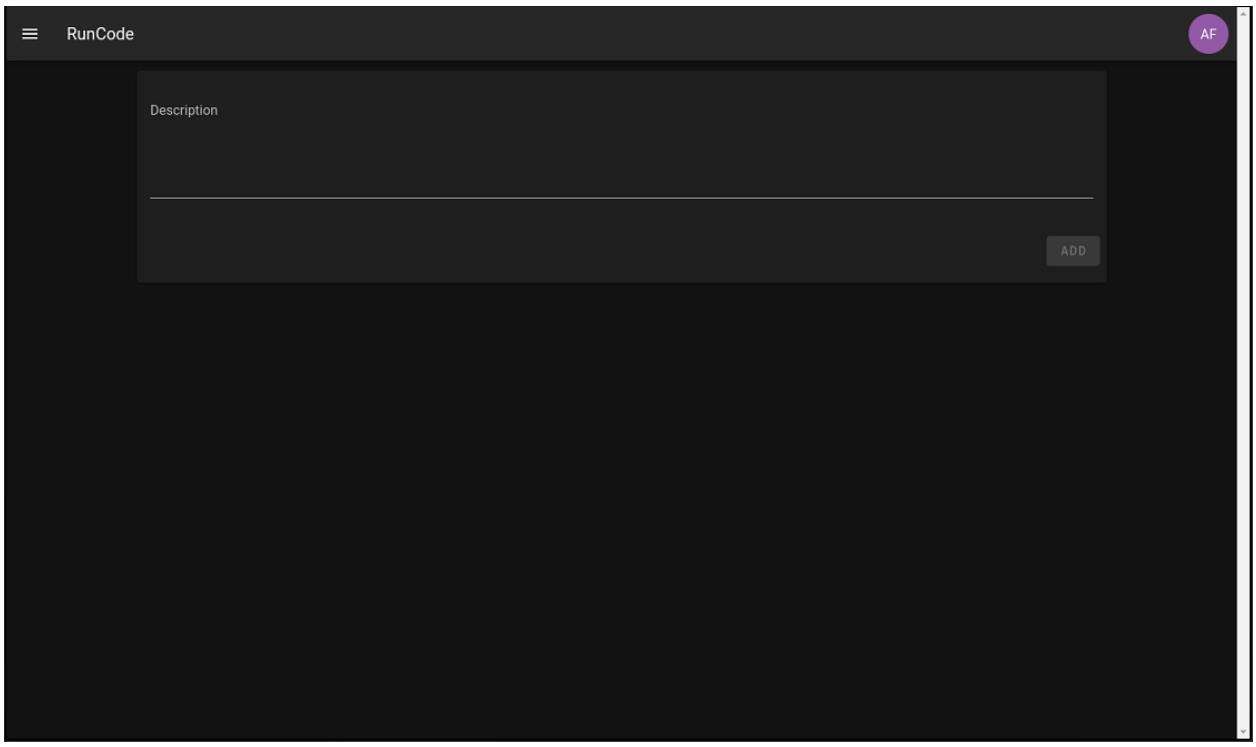


Рис. 4.10. Сторінка для створення завдання студентам

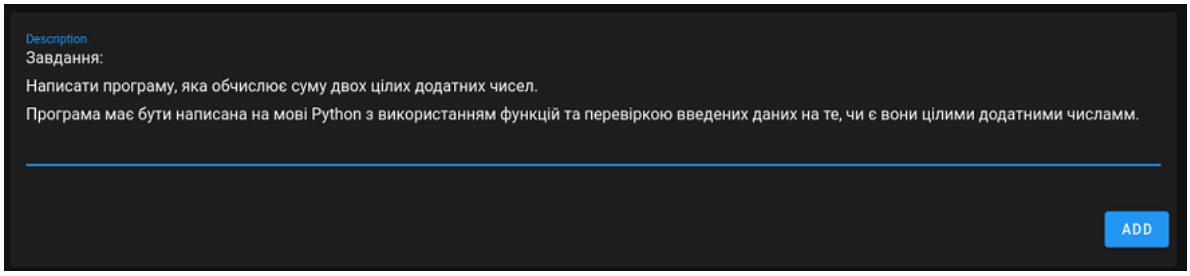


Рис. 4.11. Приклад заповненого завдання

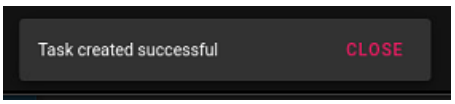


Рис. 4.12. Сповіщення про вдале додавання завдання

Завдання, що додаються викладачем бачать всі студенти, яких цей викладач додав до своєї бази даних, щоб це зробити йому спочатку необхідно перейти на сторінку пошуку та додавання зареєстрованих студентів (рис.4.13) натиснувши на вкладку «Add Student» в головному меню, або на відповідне посилання в боковому меню. Дана сторінка містить два поля для вводу даних, це ім’я (FirstName) та прізвище (LastName) студента.

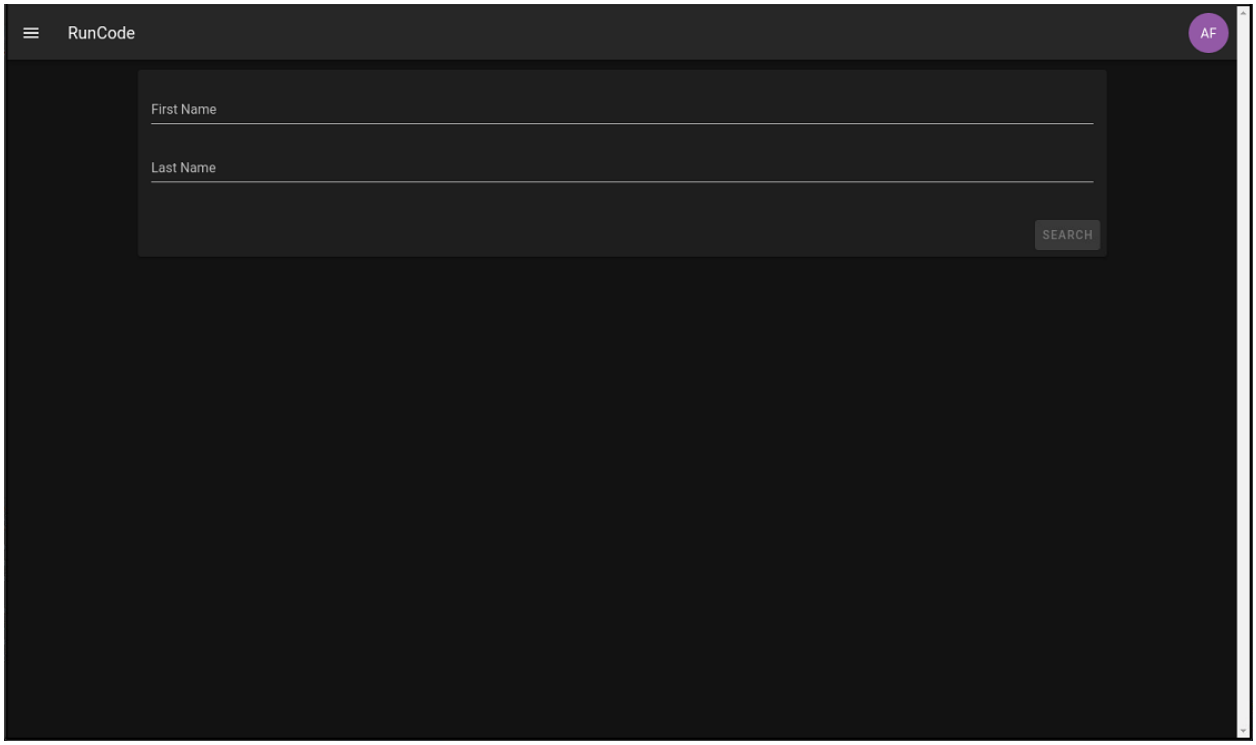


Рис. 4.13. Сторінка для додавання студентів до робочої групи

Щоб додати студента, викладач має знати його ім’я та прізвище англійською мовою, при цьому цей студент має бути зареєстрованим в системі під цими самими даними. Знаючи цю інформацію, викладач вводить ці дані у відповідні поля форми й натискає клавішу «SEARCH», після чого при умові, що даний студент зареєстрований в системі та не був попередньо доданий цим викладачем, на сторінці з’явиться відповідне посилання на знайденого студента з вказаними даними (рис.4.14). Натиснувши на клавішу «ADD» біля відповідного посилання, викладач додасть знайденого студента в свою базу даних.

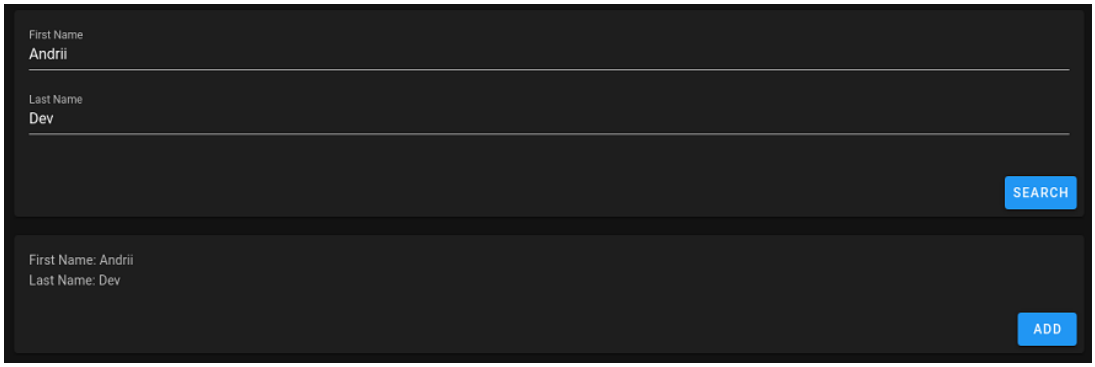


Рис. 4.14. Принцип знаходження студента в базі даних сервісу

Основною функцією для якої було розроблено даний сервіс є можливість перевірки робіт, програмних кодів студентів викладачем, цю можливість втілено на сторінці завдань (рис.4.15), яка відкривається через натискання на вкладку «Tasks» у головному меню, або в боковому меню.

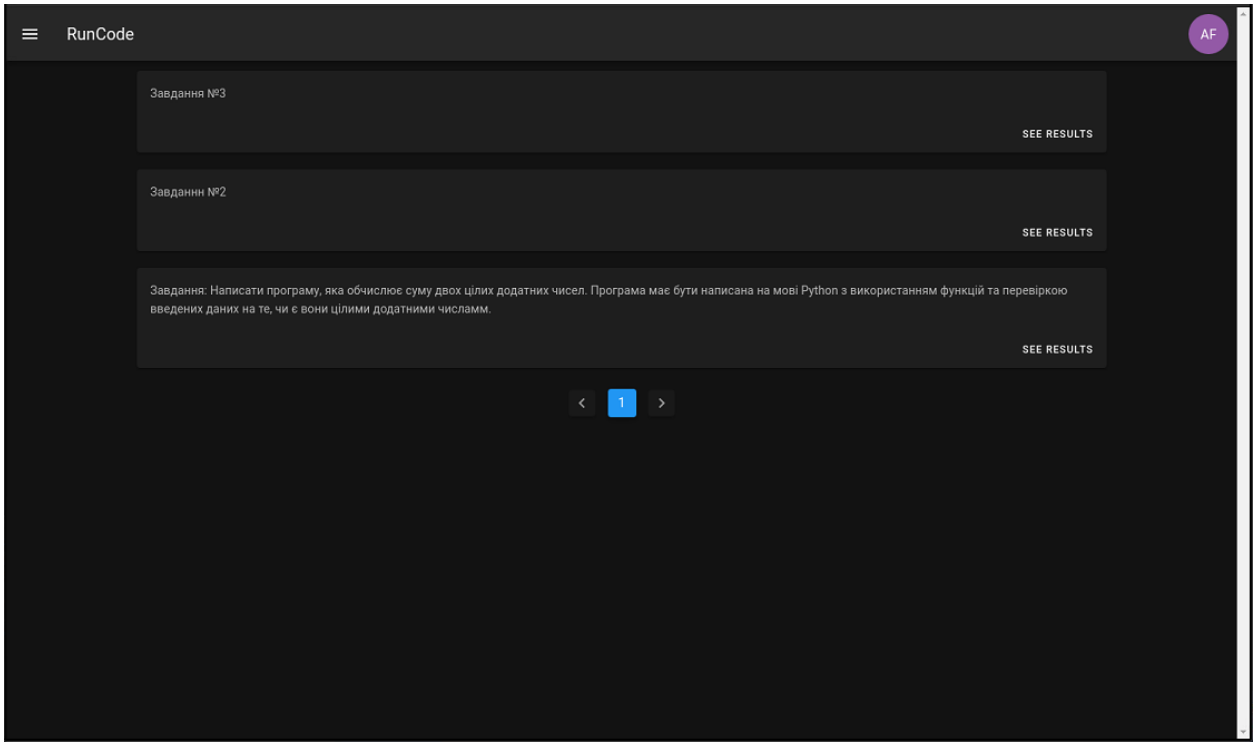


Рис. 4.15. Сторінка з завданнями

Побачити результати виконання роботи студентами (рис.4.16) можна натиснувши на кнопку «SEE RESULT» біля відповідного завдання. Вони відображаються у вигляді списку з вказуванням на те чия це відповідь, та на мову програмування, яку використовував цей студент. Ступінь перевірки

роботи визначається через колір лінії в лівій частині блоку з відповіддю студента, якщо вона сирого кольору, то ця відповідь не було перевірена, якщо вона зеленого або червоного кольору, то ця робота була перевірена і відповідно зарахована або не зарахована. Попередньо вже було створено завдання з описом, а також написана відповідь на це завдання зареєстрованим студентом, який раніше в цьому розділі був доданий до бази даних викладача.

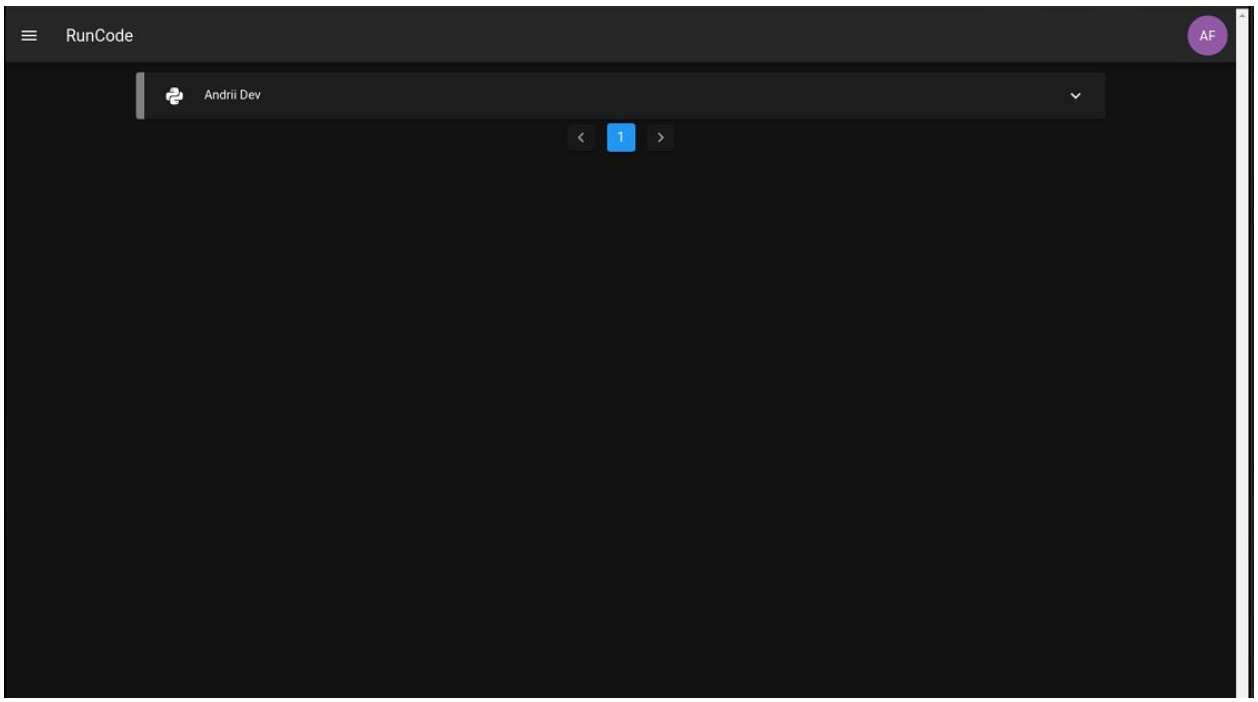


Рис. 4.16. Список відповідей студентів на завдання викладача

Натиснувши на необхідний блок, відповідь студента відобразиться у розгорнутому вигляді (рис.4.17).

Рис. 4.17. Розгорнутий вигляд відповіді на завдання

Так розгорнувши відповідь викладач бачить код програми та, одразу, результат її виконання. Він може перевірити працездатність програми з іншими даними на вже згаданій сторінці «Sandbox», після чого повернутися до перегляду відповіді або одразу оцінити роботу студента натиснувши кнопку «REVIEW» в правому нижньому куті блоку з відповіддю. У формі що відкрилася (рис.4.18) викладач може залишити свій коментар з приводу виконаної написаної студентом програми, наприклад написати оцінку, або вказати на недоліки чи помилки (рис.4.19) та натиснути одну з клавіш «DECLINE» або «ACCEPT», тобто відмовити у прийнятті або прийняти роботу студента, після чого студент зможе побачити результат перевірки.

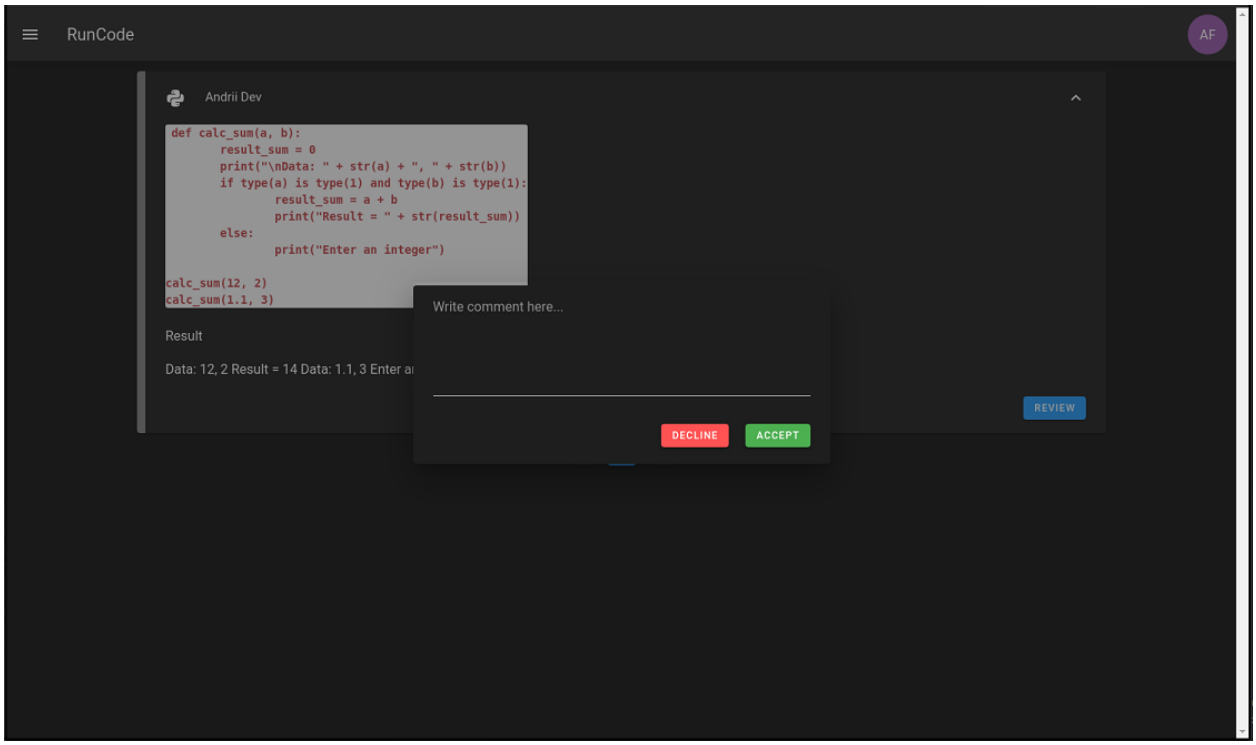


Рис. 4.18. Форма для оцінювання роботи студента викладачем

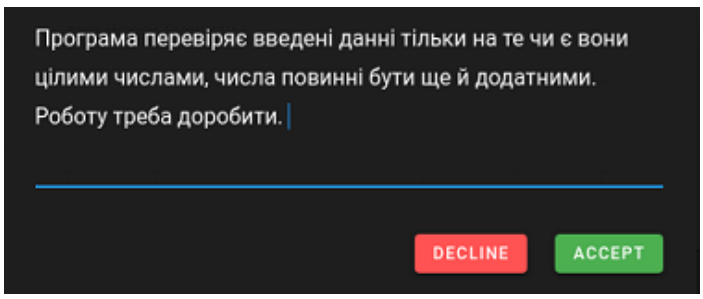


Рис. 4.19. Приклад коментаря викладача

В залежності від статусу прийнятої роботи, в результатах вона буде відмічена зеленим або червоним кольором, відповідно до того зарахована вона чи ні.

4.2. Демонстрація роботи студента в системі

В системі було зареєстровано обліковий запис студента з даними, які вказано на рис.4.20. Як можна побачити по формі різниця між реєстрацією студента та викладача відрізняється тільки обранням відповідного значення ролі.

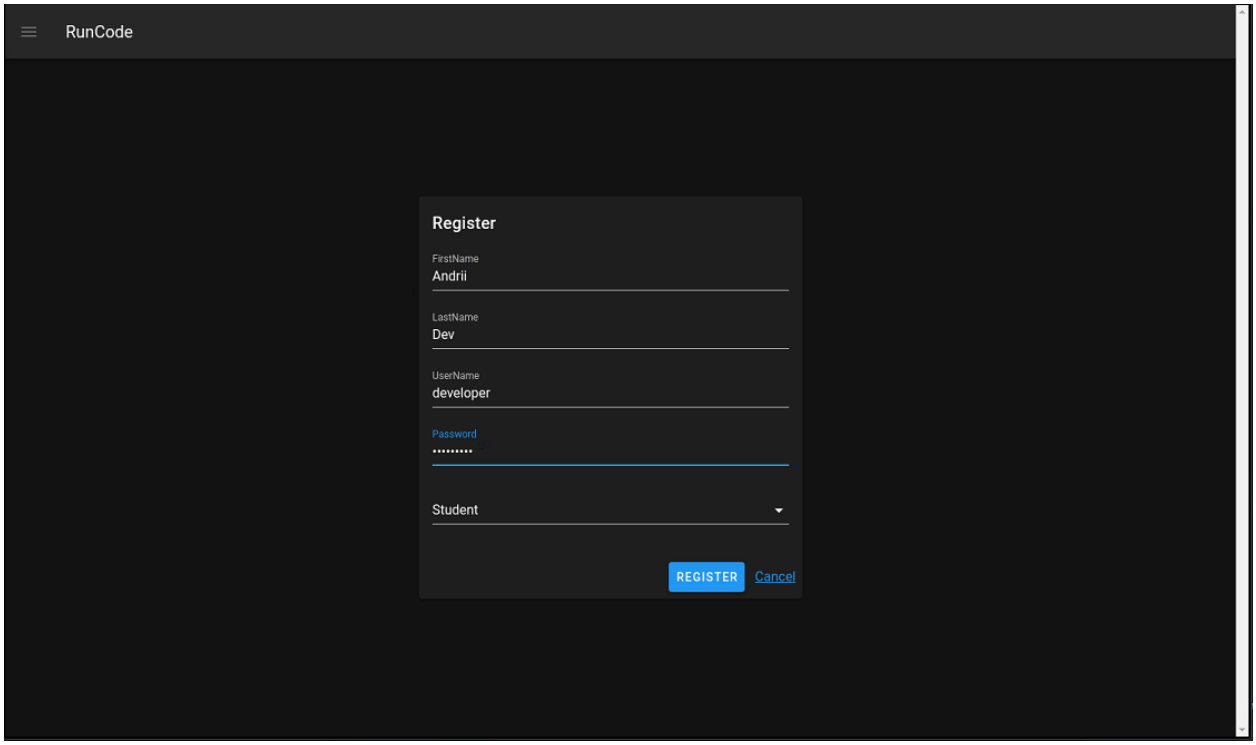


Рис. 4.20. Форма для реєстрації студента

Головна сторінка для студента (рис. 4.21) відрізняється наявністю тільки двох вкладок на відміну від чотирьох для викладача, це вкладки «Sandbox» і «Tasks», зумовлено це тим, що у студента не має необхідності в створенні завдань та груп.

Вкладка «Sandbox» працює так само як і для викладача, а сторінка яка відкривається з її допомогою має ті ж функції, що були вказані раніше для викладача, також, тими самими властивостями володіє бокове меню.

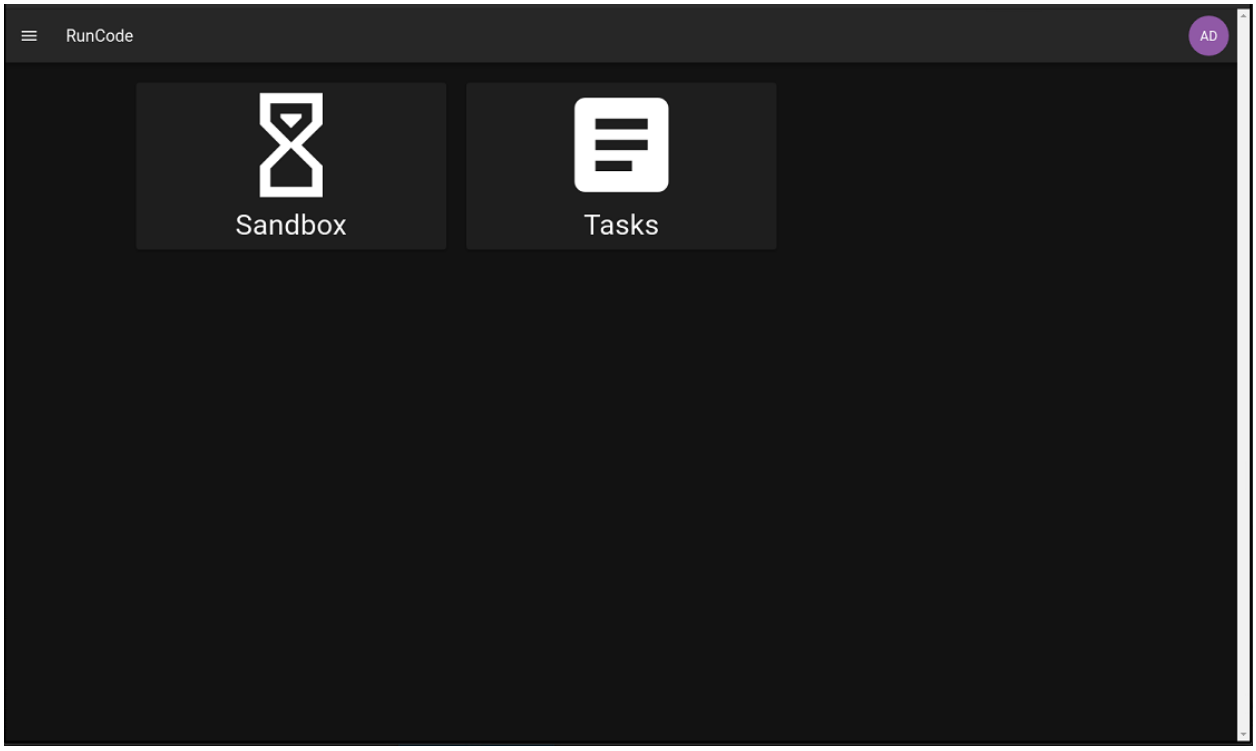


Рис. 4.21. Головна сторінка користувача «Студент»

При натисканні на вкладку «Tasks» студент бачить всі завдання, які були додані його вчителями, виглядає це як список завдань так само як і для викладача (див. рис. 4.15). Щоб подивитися результати та відповісти на завдання необхідно натиснути на кнопку «SEE RESULT» біля відповідного завдання, при цьому студент, на відміну від вчителя, бачить тільки свої відповіді, тобто не може скористатися результатами інших, як можна побачити на рис. 4.22 список відповідей пустий, оскільки студент ще нічого не надіслав, при цьому він може додавати відповідь, розгорнувши вкладку «Add result» (рис. 4.23).

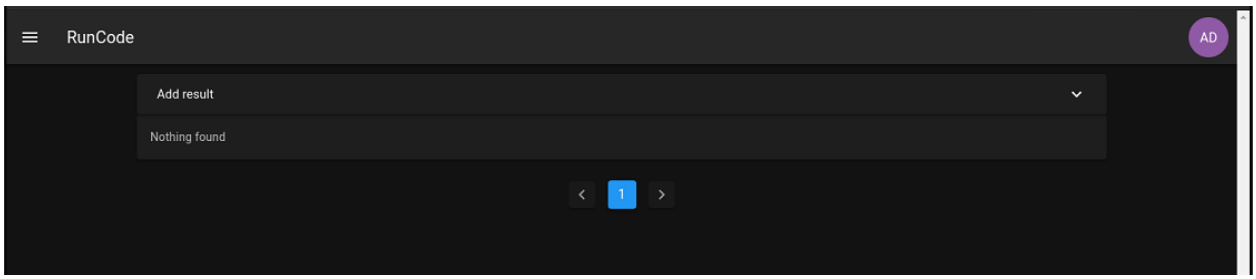


Рис. 4.22. Місце додавання відповідей

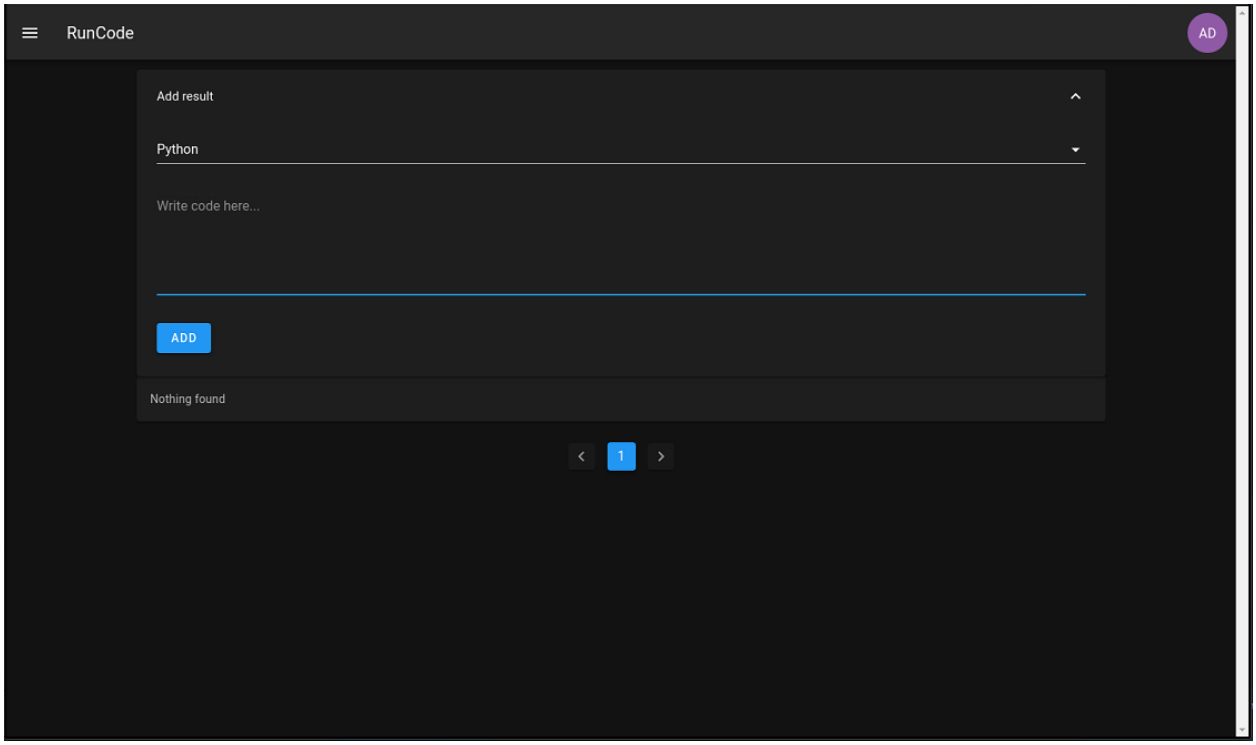


Рис. 4.23. Форма для відповіді на завдання викладача

При заповненні форми студент не побачить виводу результату роботи програми, тому бажано спочатку перевірити працездатність коду на сторінці «Sandbox» і тільки потім вставляти цей код у відповідне поле форми (рис. 4.24), також необхідно прослідкувати, щоб мова програмування була обрана правильно, в іншому разі система вважатиме програму помилковою. Додати відповідь можна за допомогою на жаття на кнопку «ADD» в лівому нижньому куті форми.

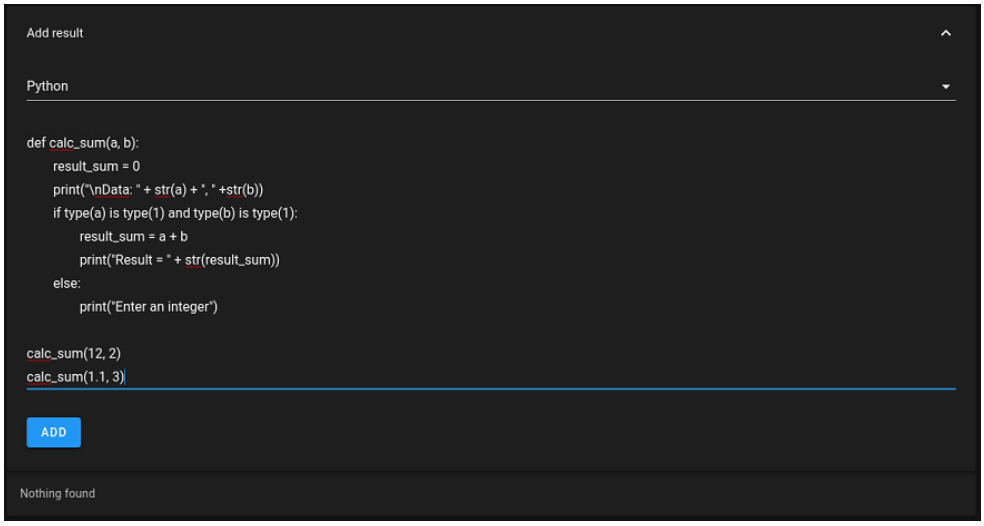


Рис. 4.24. Приклад заповнення форми

Після додавання своєї програми студент чекає відповіді викладача. Як вже було сказано в цьому розділі, якщо викладач оцінив роботу то блок з відповіддю буде відмічено зеленою або червоною лінією, відповідно при успішній чи не успішній здачі, якщо ж вона ще не була оцінена то лінія буде сірою. Якщо вчитель оцінив роботу, студент може розгорнути блок зі своєю відповіддю (рис. 4.25) і подивитися коментар, який залишив викладач, це може бути оцінка за роботу, дозвіл на здачу теорії, або вказівки на помилки в програмі, коментар розміщується у нижній частині блока.

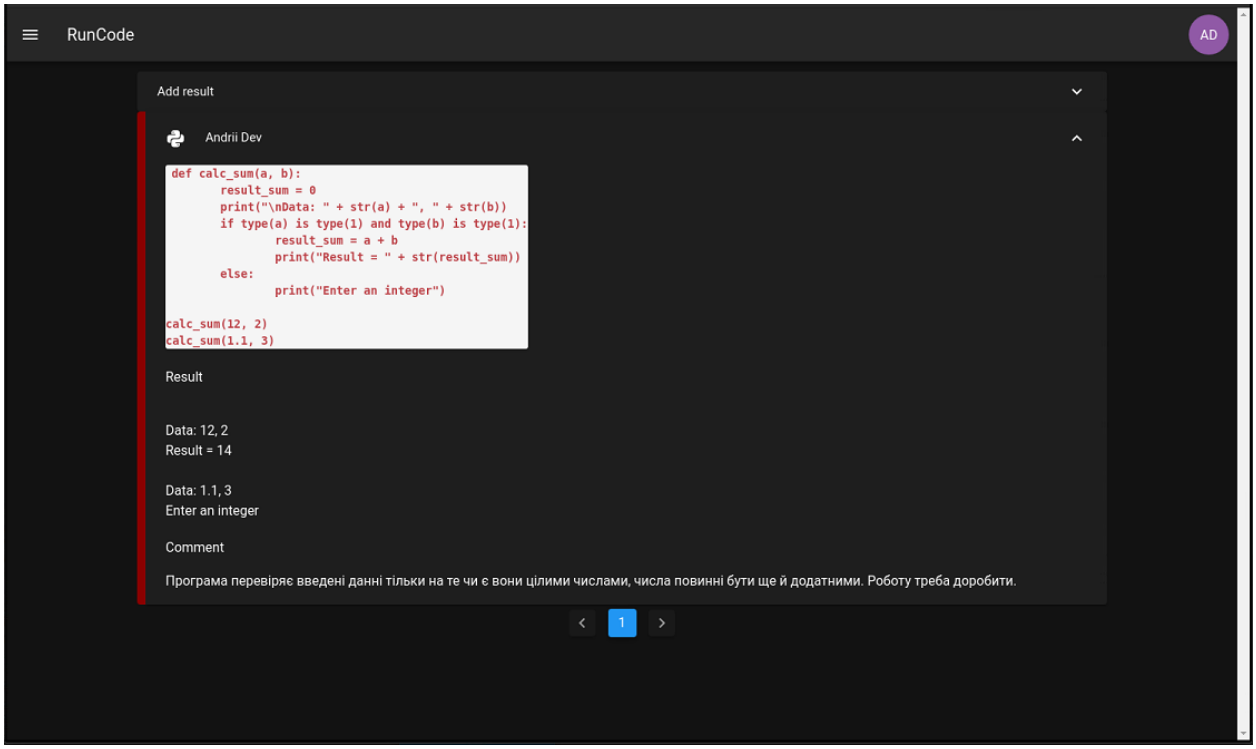


Рис. 4.25. Приклад результату перевірки роботи студента

При невдалій здачі роботи, студент може додавати нові відповіді, тим самим виправляючи старі помилки.

Висновки до розділу 4

В даному розділі було продемонстровано роботу розробленого сервісу, який був спроектований для втілення системи для перевірки лабораторних робіт студентів з програмування.

Демонстрація включає в себе всі інструкції для користувача, та включає необхідні для розуміння знімки екрану, що дозволяє розібратися зі всіма можливостями що надає система, та приступити до роботи.

Даний сервіс має закінчений та інтуїтивно зрозумілий інтерфейс користувача, що надає всю необхідну інформацію для роботи в системі. Візуальний стиль було обрано зробити в темних тонах для зменшення навантаження на зір користувача.

Інформація надана в даному розділі затверджує працездатність системи і дозволяє стверджувати, про можливість її використання в навчальних закладах для покращення навчального процесу у тих дисциплінах де основне навантаження припадає на створення коду на мовах програмування типу Python, C# та Java.

ВИСНОВКИ

Метою даної дипломної роботи було створення системи для перевірки лабораторних робіт студентів з програмування. Під час її розробки було проаналізовано велику кількість матеріалу, та використано різне програмне забезпечення для досягнення поставленої мети, тому при завершенні роботи над проектом було зроблено наступні висновки:

1. Навчальні заклади потребують удосконалення системи оцінювання робіт студентів, адже частіше за все існуючі системи є застарілими або просто не актуальними, тому часто виникає ситуація коли перевірка однієї роботи з програмування одного студента займає більшу частину часу на занятті, яку можна було б приділити іншим студентам, що зумовлено перевіркою як коду програми так і теоретичних знань студента. Постає необхідність у впровадженні в навчальний процес додаткових систем, які б допомогли розвитку дистанційного навчання, що дозволило б зменшити час здачі однієї роботи студента під час заняття, тим самим надаючи можливість перевірити знання та навички всіх студентів протягом одного заняття. Тобто викладач може перевірити програму студента дистанційно, а на занятті приділити більшу увагу теоретичним навичкам.
2. Зроблений аналіз сервісів для перевірки працездатності програмного коду та систем дистанційного навчання, показав, що головними двома проблемами при їх впровадженні в навчальний процес є неповну чи навпаки надмірну функціональність або необхідність у грошових витратах. Так одні сервіси не мають можливості для зв’язку студента та викладача, що робить такий сервіс даремним, інші ж мають повний набір необхідних функцій, але при цьому потребують грошового вкладення від студента або викладача, що є неприпустимим.
3. На основі проаналізованих систем, розроблено функціональну характеристику, яка включає всі необхідні можливості для зв’язку

студента та викладача, можливістю надання завдання викладачем, можливістю написання та запуску коду в даній системі, та нарешті перевірки і оцінювання лабораторної роботи студента викладачем.

4. Створено веб-сервіс, що втілює розроблену функціональність та має зручний інтерфейс для роботи користувача, при цьому не має необхідності у використанні додаткового програмного забезпечення зі сторони викладача чи студента. Використане при розробці програмне забезпечення є доступним та зручним у використанні, що дає можливість розвивати та доповнювати систему в майбутньому.

ПЕРЕЛІК ПОСИЛАНЬ

1. Подборка онлайн компиляторов [Електронний ресурс] – Режим доступу до ресурсу: https://tproger.ru/digest/compile-code-online/
2. Frequently asked questions IdeOne [Електронний ресурс] – Режим доступу до ресурсу: https://ideone.com/faq
3. JSFiddle Docs & Help [Електронний ресурс] – Режим доступу до ресурсу: https://docs.jsfiddle.net/
4. What is JDoodle.com [Електронний ресурс] – Режим доступу до ресурсу: https://docs.jdoodle.com/jdoodle-online-compiler-and-ide/untitled-1
5. Guru - Assignments [Електронний ресурс] – Режим доступу до ресурсу: https://docs.jdoodle.com/guru/assignments
6. Welcome to Browxy [Електронний ресурс] – Режим доступу до ресурсу: https://www.browxy.com/#
7. Что такое Learning Management System (LMS) и как с ее помощью управлять обучением [Електронний ресурс] – Режим доступу до ресурсу: https://www.ispring.ru/elearning-insights/chto-takoe-lms.
8. Moodledocs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.moodle.org/38/en/Main_page.
9. C# 6.0 draft specification [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction
10. Введение в JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/intro
11. Vue.js Guide [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/v2/guide/
12. Introduction to ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1

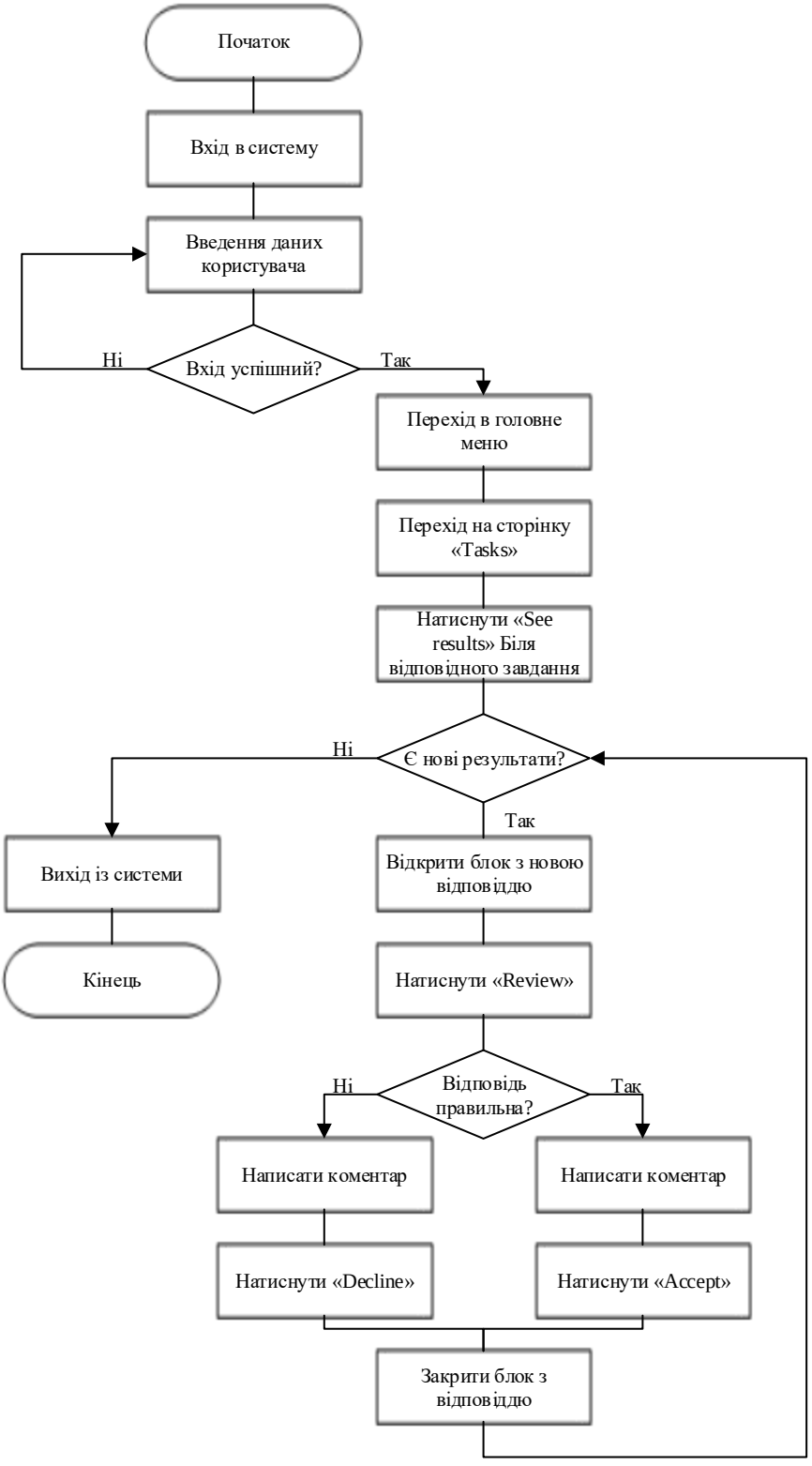
13. PostgreSQL 13beta1 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/files/documentation/pdf/13/postgresql-13-A4.pdf
14. Docker concepts [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/get-started/
15. Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-gb/ef/core/
16. What is Vuex? [Електронний ресурс] – Режим доступу до ресурсу: https://vuex.vuejs.org/
17. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/introduction/

Додаток А

Принципова схема алгоритму перевірки лабораторних робіт студентів

до дипломного проєкту
на тему: «Система перевірки лабораторних робіт студентів з програмування»

Київ – 2020 року

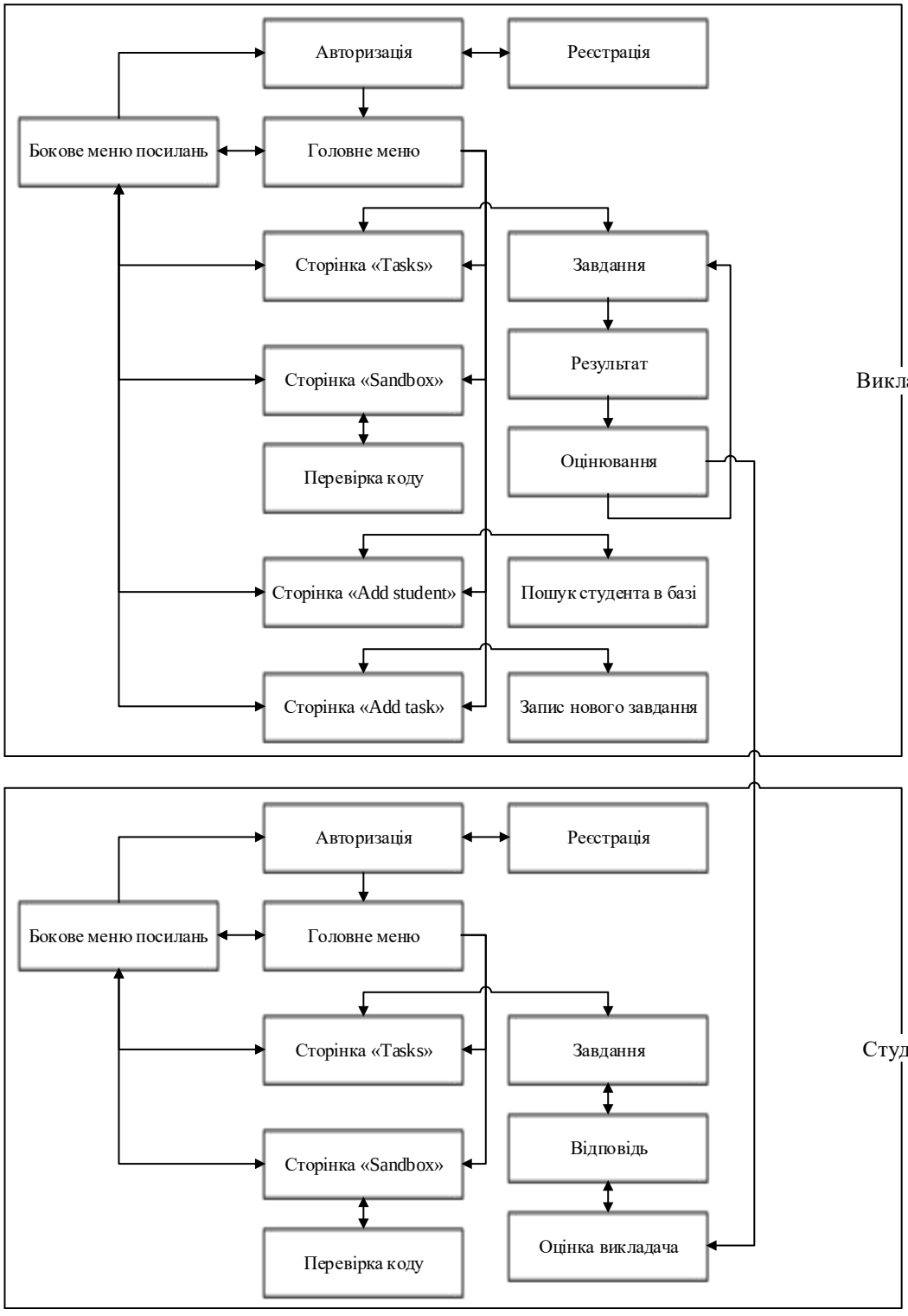


					ІАЛЦ.467100.004 Д1		
					Система перевірки лабораторних робіт студентів з програмування	*Літ.*	*Маса.*	*Масш.*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Федосов А. О.*			*Принципова схема алгоритму перевірки лабораторних робіт студентів*			
Перевір.		*Новотарський М.*				*Аркуш* 1	*Аркушів* 1	
Т. Конто.								
Н. Контр.		*Сімоненко В.П.*				*НТУУ «КПІ», ФІОТ ІО-64*		
Затверд.								

Додаток Б
Структурна схема сервісу для перевірки лабораторних робіт студентів

до дипломного проєкту
на тему: «Система перевірки лабораторних робіт студентів з програмування»

Київ – 2020 року



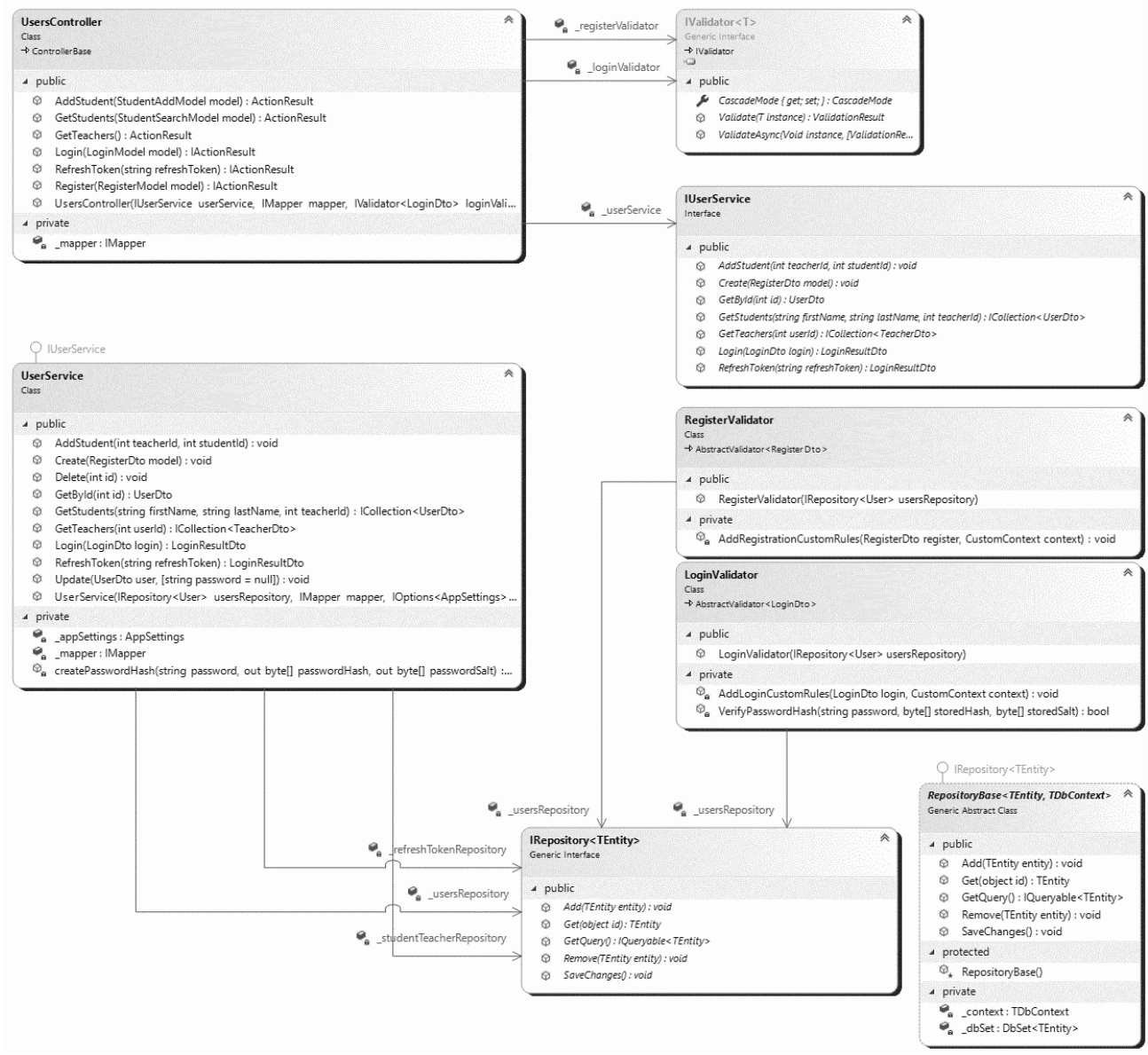
					ІАЛЦ.467100.005 Д2		
					Система перевірки лабораторних робіт студентів з програмування	*Літ.*	*Маса.*	*Масш.*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Федосов А. О.*			*Структурна схема сервісу для перевірки лабораторних робіт студентів*	*Аркуш* 1	*Аркушів* 1	
Перевір.		*Новотарський М.*						
Т. Конто.								
Н. Контр.		*Сімоненко В.П.*				*НТУУ «КПІ», ФІОТ ІО-64*		
Затверд.								

Додаток В

Функціональна схема класів системи для перевірки лабораторних робіт студентів

до дипломного проєкту
на тему: «Система перевірки лабораторних робіт студентів з програмування»

Київ – 2020 року



					ІАЛЦ.467100.006 Д3	
Змн.	Арк.	№ докум.	Підпис	Дата	Система перевірки лабораторних робіт студентів з програмування Функціональна схема класів системи для перевірки лабораторних робіт студентів	Літ. / Маса. / Масш.
Розроб.		Федосов А. О.				
Перевір.		Новотарський М.				Аркуш 1 / Аркушів 1
Т. Конто.						
Н. Контр.		Сімоненко В.П.				НТУУ «КПІ», ФІОТ ІО-64
Затверд.						

Додаток Г

до дипломного проєкту
на тему: «Система перевірки лабораторних робіт студентів з програмування»

Київ – 2020 року

ЛІСТИНГ ПРОГРАМИ (СЕРВЕРНА ЧАСТИНА)

Вміст директорії WebApplicationBLL

```
using System;
using System.Diagnostics;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using WebApplicationBLL.Interfaces;
using WebApplicationDAL.Enums;

namespace WebApplicationBLL.Services
{
    public class CodeService : ICodeService
    {
        public string Run(string command, LanguageType languageId)
        {
            var (directoryPath, fileName) = CreateCommandFile(command, languageId);
            CreateDockerFile(directoryPath, fileName, languageId);
            var imageId = RunDockerImageBuildScript(directoryPath);
            var result = RunDockerContainer(imageId);
            RemoveDockerImage(imageId);
            Directory.Delete(directoryPath, true);
            return result;
        }

        private void CreateDockerFile(string directoryPath, string fileName,
LanguageType languageId)
        {
            const string dockerFileName = "dockerfile";
            var dockerFilePath = $"{directoryPath}/{dockerFileName}";
            var dockerFile = languageId switch
            {
                LanguageType.CSharp => $@"
                        FROM mono
                        WORKDIR /home
                        COPY {fileName}.cs  .
                        RUN mcs /home/{fileName}.cs
                        CMD [""mono"", ""/home/{fileName}.exe""]",
                LanguageType.Java => $@"
                        FROM openjdk:7
                        WORKDIR /home
                        COPY Main.java  .
                        RUN javac /home/Main.java
                        CMD [""java"", ""Main""]",
                _ => $@"
                        FROM python
                        WORKDIR /home
                        COPY {fileName} .
                        CMD python /home/{fileName}"
            };
            using var dockerFileStream = File.Create(dockerFilePath);
            var dockerFileBytes = new UTF8Encoding(true).GetBytes(dockerFile);
            dockerFileStream.Write(dockerFileBytes, 0, dockerFileBytes.Length);
        }

        private (string directoryPath, string fileName) CreateCommandFile(string
command, LanguageType languageId)
```

```
        {
            var directoryName = Guid.NewGuid().ToString();
            var directoryPath =
Directory.CreateDirectory($"{Environment.CurrentDirectory}/{directoryName}");
            var fileName = Guid.NewGuid().ToString();
            var filePath = languageId switch
            {
                LanguageType.CSharp => $"{directoryPath}/{fileName}.cs",
                LanguageType.Java => $"{directoryPath}/Main.java",
                _ => $"{directoryPath}/{fileName}"
            };
            using var fileStream = File.Create(filePath);
            var fileBytes = new UTF8Encoding(true).GetBytes(command);
            fileStream.Write(fileBytes, 0, fileBytes.Length);
            return (directoryPath.ToString(), fileName);
        }

        private string RunDockerImageBuildScript(string directoryPath)
        {
            var imageId = Guid.NewGuid().ToString();
            var process = GetProcess($"cd {directoryPath} && docker build -t {imageId}
.");
            process.Start();
            // var result = process.StandardOutput.ReadToEnd();
            process.WaitForExit();
            return imageId;
        }

        private string RunDockerContainer(string imageId)
        {
            var timeout = new TimeSpan(0,0, 20);
            var task = Task.Run(() =>
            {
                var process = GetProcess($"docker run --rm --name {imageId}
{imageId}");
                process.Start();
                var result = process.StandardOutput.ReadToEnd();
                var errorResult = process.StandardError.ReadToEnd();
                process.WaitForExit();
                return $"{result}{errorResult}";
            });
            if (task.Wait(timeout))
                return task.Result;
            var containerStopProcess = GetProcess($"docker container stop {imageId}");
            containerStopProcess.Start();
            containerStopProcess.WaitForExit();
            return "Timed out";
        }

        private void RemoveDockerImage(string imageId)
        {
            var process = GetProcess($"docker rmi {imageId}");
            process.Start();
            // var result = process.StandardOutput.ReadToEnd();
            process.WaitForExit();
        }

        private Process GetProcess(string command) =>
            new Process
            {
                StartInfo = new ProcessStartInfo
                {
                    FileName = "/bin/bash",
                    Arguments = $"-c \"{command}\"",
                    RedirectStandardOutput = true,
                    RedirectStandardError = true,
                    UseShellExecute = false,
                    CreateNoWindow = true,
                }
```

```
            };
        }
    }


    using System.Collections.Generic;
    using System.Linq;
    using WebApplicationBLL.DTOs;
    using WebApplicationBLL.Interfaces;
    using WebApplicationDAL.Entities;
    using WebApplicationDAL.Enums;
    using WebApplicationDAL.Interfaces;

    namespace WebApplicationBLL.Services
    {
        public class TaskResultService : ITaskResultService
        {
            private readonly IRepository<TaskResult> _taskResultRepository;

            public TaskResultService(IRepository<TaskResult> taskResultRepository)
            {
                _taskResultRepository = taskResultRepository;
            }

            public void Create(TaskResultCreateDto taskResult)
            {
                _taskResultRepository.Add(new TaskResult
                {
                    UserId = taskResult.UserId,
                    TaskId = taskResult.TaskId,
                    Code = taskResult.Code,
                    Result = taskResult.Result,
                    TypeId = TaskResultType.NotViewed,
                    LanguageId = taskResult.LanguageId
                });
                _taskResultRepository.SaveChanges();
            }

            public TasksResultDto GetTeacherTaskResults(int taskId, int teacherId, int
    page = 1)
            {
                return new TasksResultDto
                {
                    TotalCount = _taskResultRepository.GetQuery()
                        .Count(tr => tr.TaskId == taskId
                                     && tr.Task.UserId == teacherId),
                    Results = _taskResultRepository.GetQuery().Where(tr => tr.TaskId ==
    taskId
                                                                           &&
    tr.Task.UserId == teacherId)
                        .OrderByDescending(t => t.Id)
                        .Skip((page - 1) * 10)
                        .Take(10)
                        .Select(tr => new TaskResultDto
                        {
                            Code = tr.Code,
                            FirstName = tr.User.FirstName,
                            LastName = tr.User.LastName,
                            Result = tr.Result,
                            TaskId = tr.TaskId,
                            UserId = tr.UserId,
                            TypeId = tr.TypeId,
                            LanguageId = tr.LanguageId,
                            TaskResultId = tr.Id,
                            Comment = tr.Comment
                        })
                        .ToList()
                };
            }
```

```
        public TasksResultDto GetStudentTaskResults(int taskId, int studentId, int
page = 1)
        {
            return new TasksResultDto
            {
                TotalCount = _taskResultRepository.GetQuery()
                    .Count(tr => tr.TaskId == taskId
                                 && tr.UserId == studentId),
                Results = _taskResultRepository.GetQuery().Where(tr => tr.TaskId ==
taskId
                                                                       && tr.UserId ==
studentId)
                    .OrderByDescending(t => t.Id)
                    .Skip((page - 1) * 10)
                    .Take(10)
                    .Select(tr => new TaskResultDto
                    {
                        Code = tr.Code,
                        FirstName = tr.User.FirstName,
                        LastName = tr.User.LastName,
                        Result = tr.Result,
                        TaskId = tr.TaskId,
                        UserId = tr.UserId,
                        TypeId = tr.TypeId,
                        LanguageId = tr.LanguageId,
                        Comment = tr.Comment
                    })
                    .ToList()
            };
        }

        public void SetTaskResult(int taskResultId, int teacherId, TaskResultType
type, string comment)
        {
            var taskResult = _taskResultRepository.GetQuery()
                .Where(t => t.Task.UserId == teacherId && t.Id == taskResultId
                    && t.TypeId == TaskResultType.NotViewed)
                .ToList()
                .First();

            taskResult.TypeId = type;
            taskResult.Comment = comment;

            _taskResultRepository.SaveChanges();
        }
    }
}


using System.Collections.Generic;
using System.Linq;
using WebApplicationBLL.DTOs;
using WebApplicationBLL.Interfaces;
using WebApplicationDAL.Entities;
using WebApplicationDAL.Interfaces;

namespace WebApplicationBLL.Services
{
    public class TaskService : ITaskService
    {
        private readonly IRepository<Task> _taskRepository;
        private readonly IRepository<StudentTeacher> _studentTeacherRepository;

        public TaskService(IRepository<Task> taskRepository,
IRepository<StudentTeacher> studentTeacherRepository)
        {
            _taskRepository = taskRepository;
            _studentTeacherRepository = studentTeacherRepository;
```

```
        }

        public void Create(TaskCreateDto task)
        {
            _taskRepository.Add(new Task
            {
                Description = task.Description,
                UserId = task.UserId
            });
            _taskRepository.SaveChanges();
        }

        public TasksDto GetUserTasks(int studentId, int page = 1)
        {
            var teacherIds = _studentTeacherRepository.GetQuery()
                .Where(s => s.StudentId == studentId)
                .Select(s => s.TeacherId)
                .ToList();
            var tasksCount = _taskRepository
                .GetQuery()
                .Count(t => teacherIds.Contains(t.UserId));
            return new TasksDto
            {
                TotalCount = tasksCount,
                Tasks = _taskRepository.GetQuery()
                    .Where(t => teacherIds.Contains(t.UserId))
                    .OrderByDescending(t => t.Id)
                    .Skip((page - 1) * 10)
                    .Take(10)
                    .Select(t => new TaskDto
                    {
                        Id = t.Id,
                        Description = t.Description
                    }).ToList()
            };
        }

        public TasksDto GetTeacherTasks(int teacherId, int page = 1)
        {
            var tasksCount = _taskRepository
                .GetQuery()
                .Count(t => t.UserId == teacherId);
            return new TasksDto
            {
                TotalCount = tasksCount,
                Tasks = _taskRepository.GetQuery()
                    .Where(t => t.UserId == teacherId)
                    .OrderByDescending(t => t.Id)
                    .Skip((page - 1) * 10)
                    .Take(10)
                    .Select(t => new TaskDto
                    {
                        Id = t.Id,
                        Description = t.Description
                    }).ToList()
            };
        }
    }
}


using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;
using AutoMapper;
```

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using WebApplicationBLL.DTOs;
using WebApplicationBLL.Helpers;
using WebApplicationBLL.Interfaces;
using WebApplicationDAL.Entities;
using WebApplicationDAL.Interfaces;

namespace WebApplicationBLL.Services
{
    public class UserService : IUserService
    {
        private readonly IRepository<User> _usersRepository;
        private readonly IRepository<RefreshToken> _refreshTokenRepository;
        private readonly IMapper _mapper;
        private readonly AppSettings _appSettings;
        private readonly IRepository<StudentTeacher> _studentTeacherRepository;

        public UserService(IRepository<User> usersRepository,
            IMapper mapper,
            IOptions<AppSettings> appSettings, IRepository<RefreshToken>
refreshTokenRepository,
            IRepository<StudentTeacher> studentTeacherRepository)
        {
            _usersRepository = usersRepository;
            _mapper = mapper;
            _refreshTokenRepository = refreshTokenRepository;
            _studentTeacherRepository = studentTeacherRepository;
            _appSettings = appSettings.Value;
        }

        public LoginResultDto Login(LoginDto login)
        {
            var user = _usersRepository
                .GetQuery()
                .AsNoTracking()
                .First(x => x.Username == login.UserName);
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(_appSettings.Secret);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new[]
                {
                    new Claim(ClaimTypes.Name, user.Id.ToString()),
                    new Claim(ClaimTypes.Role, user.Role),
                }),
                //Expires = DateTime.UtcNow.AddMinutes(1),
                Expires = DateTime.UtcNow.AddDays(30),
                SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key),
                    SecurityAlgorithms.HmacSha256Signature)
            };
            var refreshToken = new RefreshToken
            {
                     UserId = user.Id,
                     Token = Guid.NewGuid().ToString()
            };
            _refreshTokenRepository.Add(refreshToken);
            _refreshTokenRepository.SaveChanges();
            var token = tokenHandler.CreateToken(tokenDescriptor);
            var tokenString = tokenHandler.WriteToken(token);
            return new LoginResultDto
            {
                Id = user.Id,
                Username = user.Username,
                FirstName = user.FirstName,
                LastName = user.LastName,
                Token = tokenString,
```

```
                RefreshToken = refreshToken.Token,
                Role = user.Role
            };
        }

        public LoginResultDto RefreshToken(string refreshToken)
        {
                var result = _refreshTokenRepository
                        .GetQuery()
                .Include(r => r.User)
                        .First(r => r.Token == refreshToken);
                var tokenHandler = new JwtSecurityTokenHandler();
                var key = Encoding.ASCII.GetBytes(_appSettings.Secret);
                var tokenDescriptor = new SecurityTokenDescriptor
                {
                        Subject = new ClaimsIdentity(new[]
                        {
                                new Claim(ClaimTypes.Name, result.UserId.ToString()),
                    new Claim(ClaimTypes.Role, result.User.Role),
                        }),
                        //Expires = DateTime.UtcNow.AddMinutes(1),
                Expires = DateTime.UtcNow.AddDays(30),
                SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key),
                                SecurityAlgorithms.HmacSha256Signature)
                };

                result.Token = Guid.NewGuid().ToString();
                _refreshTokenRepository.SaveChanges();
                var token = tokenHandler.CreateToken(tokenDescriptor);
                var tokenString = tokenHandler.WriteToken(token);
                return new LoginResultDto
                {
                        Token = tokenString,
                        RefreshToken = result.Token
                };
        }

        public UserDto GetById(int id)
        {
            var user = _usersRepository.GetQuery()
                .FirstOrDefault(u => u.Id == id);
            return _mapper.Map<User, UserDto>(user);
        }

        public void Create(RegisterDto model)
        {
            var user = _mapper.Map<RegisterDto, User>(model);
            createPasswordHash(model.Password, out var passwordHash, out var
passwordSalt);
            user.PasswordHash = passwordHash;
            user.PasswordSalt = passwordSalt;
            _usersRepository.Add(user);
            _usersRepository.SaveChanges();
        }

        public void Update(UserDto user, string password = null)
        {
            throw new NotImplementedException();
        }

        public void Delete(int id)
        {
            throw new NotImplementedException();
        }

        public ICollection<TeacherDto> GetTeachers(int userId)
        {
            return _studentTeacherRepository.GetQuery()
```

```
                .Where(st => st.StudentId == userId)
                .Select(st => new TeacherDto
                {
                    Id = st.Teacher.Id,
                    FirstName = st.Teacher.FirstName,
                    LastName = st.Teacher.LastName
                })
                .ToList();
        }

        public ICollection<UserDto> GetStudents(string firstName, string lastName, int
teacherId)
        {
            var users = _usersRepository.GetQuery()
                .Where(u => u.FirstName == firstName && u.LastName == lastName)
                .Select(u => new UserDto
                {
                    Id = u.Id,
                    FirstName = u.FirstName,
                    LastName = u.LastName
                })
                .ToList();
            var userIds = users.Select(u => u.Id).ToList();
            var studentIds = _studentTeacherRepository.GetQuery()
                .Where(u => u.TeacherId == teacherId && userIds.Contains(u.StudentId))
                .Select(u => u.StudentId)
                .ToList();
            return users.Where(u => !studentIds.Contains(u.Id)).ToList();
        }

        public void AddStudent(int teacherId, int studentId)
        {
            _studentTeacherRepository.Add(new StudentTeacher
            {
                TeacherId = teacherId,
                StudentId = studentId
            });
            _studentTeacherRepository.SaveChanges();
        }

        private void createPasswordHash(string password, out byte[] passwordHash, out
byte[] passwordSalt)
        {
            using var hmac = new HMACSHA512();
            passwordSalt = hmac.Key;
            passwordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
        }
    }
}


using WebApplicationDAL.Enums;

namespace WebApplicationBLL.Interfaces
{
    public interface ICodeService
    {
        string Run(string command, LanguageType languageId);
    }
}


using WebApplicationBLL.DTOs;
using WebApplicationDAL.Enums;

namespace WebApplicationBLL.Interfaces
{
    public interface ITaskResultService
    {
```

```
        void Create(TaskResultCreateDto taskResult);

        TasksResultDto GetTeacherTaskResults(int taskId, int teacherId, int page);

        TasksResultDto GetStudentTaskResults(int taskId, int studentId, int page);

        void SetTaskResult(int taskResultId, int teacherId, TaskResultType result,
string comment);
    }
}


using System.Collections.Generic;
using WebApplicationBLL.DTOs;

namespace WebApplicationBLL.Interfaces
{
    public interface ITaskService
    {
        void Create(TaskCreateDto task);

        TasksDto GetUserTasks(int studentId, int page = 1);

        TasksDto GetTeacherTasks(int teacherId, int page = 1);
    }
}


using System.Collections.Generic;
using WebApplicationBLL.DTOs;

namespace WebApplicationBLL.Interfaces
{
    public interface IUserService
    {
        LoginResultDto Login(LoginDto login);

        LoginResultDto RefreshToken(string refreshToken);

        UserDto GetById(int id);

        void Create(RegisterDto model);

        ICollection<TeacherDto> GetTeachers(int userId);

        ICollection<UserDto> GetStudents(string firstName, string lastName, int
teacherId);

        void AddStudent(int teacherId, int studentId);
    }
}


using System.Linq;
using System.Security.Cryptography;
using System.Text;
using FluentValidation;
using FluentValidation.Results;
using FluentValidation.Validators;
using Microsoft.EntityFrameworkCore;
using WebApplicationBLL.DTOs;
 using WebApplicationDAL.Entities;
 using WebApplicationDAL.Interfaces;

namespace WebApplicationBLL.Validators
{
    public class LoginValidator : AbstractValidator<LoginDto>
    {
        private readonly IRepository<User> _usersRepository;
```

```
        public LoginValidator(IRepository<User> usersRepository)
        {
            _usersRepository = usersRepository;
            RuleFor(l => l.Password).NotEmpty();
            RuleFor(l => l.UserName).NotEmpty();
            RuleFor(l => l).Custom(AddLoginCustomRules);
        }

        private void AddLoginCustomRules(LoginDto login, CustomContext context)
        {
            var user = _usersRepository
                .GetQuery()
                .AsNoTracking()
                .FirstOrDefault(x => x.Username == login.UserName);

            if (user == null)
            {
                context.AddFailure(new ValidationFailure("Username", "User does not
exist."));
                return;;
            }
            if (!VerifyPasswordHash(login.Password, user.PasswordHash,
user.PasswordSalt))
            {
                context.AddFailure(new ValidationFailure("Username", "Username or
password is invalid."));
            }
        }

        private bool VerifyPasswordHash(string password, byte[] storedHash, byte[]
storedSalt)
        {
            if (storedHash.Length != 64 || storedSalt.Length != 128)
            {
                return false;
            }
            using (var hmac = new HMACSHA512(storedSalt))
            {
                var computedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
                if (computedHash.Where((t, i) => t != storedHash[i]).Any())
                {
                    return false;
                }
            }
            return true;
        }
    }
}


using System.Linq;
using FluentValidation;
using FluentValidation.Results;
using FluentValidation.Validators;
using Microsoft.EntityFrameworkCore;
using WebApplicationDAL.Interfaces;
using WebApplicationBLL.DTOs;
using WebApplicationDAL.Entities;

namespace WebApplicationBLL.Validators
{
    public class RegisterValidator : AbstractValidator<RegisterDto>
    {
        private readonly IRepository<User> _usersRepository;

        public RegisterValidator(IRepository<User> usersRepository)
        {
            _usersRepository = usersRepository;
```

```
            RuleFor(r => r.FirstName).NotEmpty();
            RuleFor(r => r.LastName).NotEmpty();
            RuleFor(r => r.UserName).NotEmpty();
            RuleFor(r => r.Password).NotEmpty();
            RuleFor(r => r.Role).NotEmpty().Must(r => r == RoleDto.Student || r ==
RoleDto.Teacher);
            RuleFor(r => r).Custom(AddRegistrationCustomRules);
        }

        private void AddRegistrationCustomRules(RegisterDto register, CustomContext
context)
        {
            if (_usersRepository
                .GetQuery()
                .AsNoTracking()
                .Any(x => x.Username == register.UserName))
            {
                context.AddFailure(new ValidationFailure("Username", $"Username
{register.UserName} is already taken."));
            }
        }
    }
}


using FluentValidation;
using WebApplicationBLL.DTOs;

namespace WebApplicationBLL.Validators
{
    public class TaskCreateValidator : AbstractValidator<TaskCreateDto>
    {
        public TaskCreateValidator()
        {
            RuleFor(r => r.Description).NotEmpty();
            RuleFor(r => r.UserId).NotEmpty();
        }
    }
}


using FluentValidation;
using WebApplicationBLL.DTOs;

namespace WebApplicationBLL.Validators
{
    public class TaskResultCreateValidator: AbstractValidator<TaskResultCreateDto>
    {
        public TaskResultCreateValidator()
        {
            RuleFor(r => r.Code).NotEmpty();
            RuleFor(r => r.TaskId).NotEmpty();
            RuleFor(r => r.UserId).NotEmpty();
        }
    }
}


namespace WebApplicationBLL.Helpers
{
    public class AppSettings
    {
        public string Secret { get; set; }
    }
}


namespace WebApplicationBLL.DTOs
{
```

```
    public class LoginDto
    {
        public string UserName { get; set; }

        public string Password { get; set; }
    }
}


namespace WebApplicationBLL.DTOs
{
    public class LoginResultDto
    {
        public int Id { get; set; }

        public string FirstName { get; set; }

        public string LastName { get; set; }

        public string Username { get; set; }

        public string Token { get; set; }

        public string RefreshToken { get; set; }

        public string Role { get; set; }
    }
}


namespace WebApplicationBLL.DTOs
{
    public class RegisterDto : LoginDto
    {
        public string FirstName { get; set; }

        public string LastName { get; set; }

        public string Role { get; set; }
    }
}


namespace WebApplicationBLL.DTOs
{
    public class RoleDto
    {
        public const string Teacher = "Teacher";

        public const string Student = "Student";
    }
}


namespace WebApplicationBLL.DTOs
{
    public class TaskCreateDto
    {
        public string Description { get; set; }

        public int UserId { get; set; }
    }
}


using System.Collections.Generic;

namespace WebApplicationBLL.DTOs
{
```

```
    public class TasksDto {
        public IReadOnlyCollection<TaskDto> Tasks { get; set; }

        public int TotalCount { get; set; }
    }

    public class TaskDto
    {
        public int Id { get; set; }

        public string Description { get; set; }
    }
}


using WebApplicationDAL.Enums;

namespace WebApplicationBLL.DTOs
{
    public class TaskResultCreateDto
    {
        public string Code { get; set; }

        public string Result { get; set; }

        public int UserId { get; set; }

        public int TaskId { get; set; }

        public LanguageType LanguageId { get; set; }

        public int TaskResultId { get; set; }
    }
}


using WebApplicationDAL.Enums;

namespace WebApplicationBLL.DTOs
{
    public class TaskResultDto : TaskResultCreateDto
    {
        public string FirstName { get; set; }

        public string LastName { get; set; }

        public TaskResultType TypeId { get; set; }

        public string Comment { get; set; }
    }
}


using System.Collections.Generic;

namespace WebApplicationBLL.DTOs
{
    public class TasksResultDto
    {
        public IReadOnlyCollection<TaskResultDto> Results { get; set; }

        public int TotalCount { get; set; }
    }
}


namespace WebApplicationBLL.DTOs
{
    public class TeacherDto
```

```
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
    }
}


namespace WebApplicationBLL.DTOs
{
    public class UserDto
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Username { get; set; }
        public string Password { get; set; }
    }
}
```

Вміст директорії WebApplication

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using WebApplication.Models;
using WebApplicationBLL.Interfaces;

namespace WebApplication.Controllers
{
    [Authorize]
    [ApiController]
    [Route("codes")]
    public class CodeController : ControllerBase
    {
        private readonly ICodeService _codeService;

        public CodeController(ICodeService codeService)
        {
            _codeService = codeService;
        }

        [HttpPost("run")]
        public IActionResult Run(CodeModel code)
        {
            return Ok(new { Result = _codeService.Run(code.Command, code.LanguageId)
});
        }
    }
}


using System;
using System.Security.Claims;
using AutoMapper;
using FluentValidation;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using WebApplication.Models;
using WebApplicationBLL.DTOs;
using WebApplicationBLL.Interfaces;

namespace WebApplication.Controllers
{
    [ApiController]
    [Route("tasks")]
    public class TaskController : ControllerBase
```

```
    {
        private readonly ITaskService _taskService;
        private readonly IMapper _mapper;
        private readonly IValidator<TaskCreateDto> _taskCreateValidator;
        private readonly ITaskResultService _taskResultService;
        private readonly ICodeService _codeService;
        private readonly IValidator<TaskResultCreateDto> _taskResultCreateValidator;

        public TaskController(ITaskService taskService,
            IMapper mapper,
            IValidator<TaskCreateDto> taskCreateValidator,
            ITaskResultService taskResultService,
            ICodeService codeService,
            IValidator<TaskResultCreateDto> taskResultCreateValidator)
        {
            _taskService = taskService;
            _mapper = mapper;
            _taskCreateValidator = taskCreateValidator;
            _taskResultService = taskResultService;
            _codeService = codeService;
            _taskResultCreateValidator = taskResultCreateValidator;
        }

        [Authorize(Roles = RoleDto.Teacher)]
        [HttpPost("create")]
        public IActionResult CreateTask(TaskCreateModel model)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId = claimsIdentity.FindFirst(ClaimTypes.Name).Value;
            var task = _mapper.Map<TaskCreateModel, TaskCreateDto>(model);
            task.UserId = Convert.ToInt32(userId);
            var validationResult = _taskCreateValidator.Validate(task);
            if (!validationResult.IsValid)
            {
                return BadRequest(validationResult.Errors);
            }
            _taskService.Create(task);
            return Ok();
        }

        [Authorize(Roles = RoleDto.Student)]
        [HttpPost("{taskId}/result/create")]
        public IActionResult Create(int taskId, TaskResultCreateModel model)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId = claimsIdentity.FindFirst(ClaimTypes.Name).Value;
            var task = _mapper.Map<TaskResultCreateModel, TaskResultCreateDto>(model);
            task.UserId = Convert.ToInt32(userId);
            task.TaskId = taskId;
            var validationResult = _taskResultCreateValidator.Validate(task);
            if (!validationResult.IsValid)
            {
                return BadRequest(validationResult.Errors);
            }
            task.Result = _codeService.Run(task.Code, task.LanguageId);
            _taskResultService.Create(task);
            return Ok();
        }

        [Authorize(Roles = RoleDto.Student)]
        [HttpGet("user/{page}")]
        public IActionResult GetUserTasks(int page)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId =
Convert.ToInt32(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            return Ok(_taskService.GetUserTasks(userId, page));
        }
```

```
        [Authorize(Roles = RoleDto.Teacher)]
        [HttpGet("teacher/{page}")]
        public IActionResult GetTeacherTasks(int page)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId =
Convert.ToInt32(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            return Ok(_taskService.GetTeacherTasks(userId, page));
        }

        [Authorize(Roles = RoleDto.Teacher)]
        [HttpPost("teacher/{taskId}/result")]
        public IActionResult GetTeacherTaskResults(int taskId, TaskResultModel model)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId =
Convert.ToInt32(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            return Ok(_taskResultService.GetTeacherTaskResults(taskId, userId,
model.Page));
        }

        [Authorize(Roles = RoleDto.Student)]
        [HttpPost("student/{taskId}/result")]
        public IActionResult GetStudentTaskResults(int taskId, TaskResultModel model)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId =
Convert.ToInt32(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            return Ok(_taskResultService.GetStudentTaskResults(taskId, userId,
model.Page));
        }

        [Authorize(Roles = RoleDto.Teacher)]
        [HttpPost("result/{taskResultId}")]
        public IActionResult SetTaskResult(int taskResultId, TaskResultUpdateModel
model)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId =
Convert.ToInt32(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            _taskResultService.SetTaskResult(taskResultId, userId, model.Type,
model.Comment);
            return Ok();
        }
    }
}


using System;
using System.Security.Claims;
using AutoMapper;
using FluentValidation;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using WebApplication.Models;
using WebApplicationBLL.DTOs;
using WebApplicationBLL.Interfaces;

namespace WebApplication.Controllers
{
    [Authorize]
    [ApiController]
    [Route("users")]
    public class UsersController : ControllerBase
    {
        private readonly IUserService _userService;
        private readonly IValidator<LoginDto> _loginValidator;
        private readonly IValidator<RegisterDto> _registerValidator;
        private readonly IMapper _mapper;
```

```
        public UsersController(IUserService userService,
            IMapper mapper,
            IValidator<LoginDto> loginValidator,
            IValidator<RegisterDto> registerValidator)
        {
            _userService = userService;
            _mapper = mapper;
            _loginValidator = loginValidator;
            _registerValidator = registerValidator;
        }

        [AllowAnonymous]
        [HttpPost("login")]
        public IActionResult Login(LoginModel model)
        {
            var login = _mapper.Map<LoginModel, LoginDto>(model);
            var validationResult = _loginValidator.Validate(login);
            if (!validationResult.IsValid)
            {
                return BadRequest(validationResult.Errors);
            }
            var result = _userService.Login(login);
            return Ok(result);
        }

        [AllowAnonymous]
        [HttpPost("refreshToken/{refreshToken}")]
        public IActionResult RefreshToken([FromRoute]string refreshToken)
        {
                var result = _userService.RefreshToken(refreshToken);
            return Ok(result);
        }

        [AllowAnonymous]
        [HttpPost("register")]
        public IActionResult Register(RegisterModel model)
        {
            var register = _mapper.Map<RegisterModel, RegisterDto>(model);
            var validationResult = _registerValidator.Validate(register);
            if (!validationResult.IsValid)
            {
                return BadRequest(validationResult.Errors);
            }
            _userService.Create(register);
            return Ok();
        }

        [HttpGet("teachers")]
        [Authorize(Roles = RoleDto.Student)]
        public ActionResult GetTeachers()
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId = int.Parse(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            return Ok(_userService.GetTeachers(userId));
        }

        [HttpPost("students")]
        [Authorize(Roles = RoleDto.Teacher)]
        public ActionResult GetStudents(StudentSearchModel model)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId = int.Parse(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            return Ok(_userService.GetStudents(model.FirstName, model.LastName,
userId));
        }

        [HttpPost("students/add")]
        [Authorize(Roles = RoleDto.Teacher)]
```

```
        public ActionResult AddStudent(StudentAddModel model)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId = int.Parse(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            _userService.AddStudent(userId, model.StudentId);
            return Ok();
        }
    }
}


using System;
using System.Security.Claims;
using AutoMapper;
using FluentValidation;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using WebApplication.Models;
using WebApplicationBLL.DTOs;
using WebApplicationBLL.Interfaces;

namespace WebApplication.Controllers
{
    [Authorize]
    [ApiController]
    [Route("users")]
    public class UsersController : ControllerBase
    {
        private readonly IUserService _userService;
        private readonly IValidator<LoginDto> _loginValidator;
        private readonly IValidator<RegisterDto> _registerValidator;
        private readonly IMapper _mapper;

        public UsersController(IUserService userService,
            IMapper mapper,
            IValidator<LoginDto> loginValidator,
            IValidator<RegisterDto> registerValidator)
        {
            _userService = userService;
            _mapper = mapper;
            _loginValidator = loginValidator;
            _registerValidator = registerValidator;
        }

        [AllowAnonymous]
        [HttpPost("login")]
        public IActionResult Login(LoginModel model)
        {
            var login = _mapper.Map<LoginModel, LoginDto>(model);
            var validationResult = _loginValidator.Validate(login);
            if (!validationResult.IsValid)
            {
                return BadRequest(validationResult.Errors);
            }
            var result = _userService.Login(login);
            return Ok(result);
        }

        [AllowAnonymous]
        [HttpPost("refreshToken/{refreshToken}")]
        public IActionResult RefreshToken([FromRoute]string refreshToken)
        {
                var result = _userService.RefreshToken(refreshToken);
            return Ok(result);
        }

        [AllowAnonymous]
        [HttpPost("register")]
        public IActionResult Register(RegisterModel model)
```

```
        {
            var register = _mapper.Map<RegisterModel, RegisterDto>(model);
            var validationResult = _registerValidator.Validate(register);
            if (!validationResult.IsValid)
            {
                return BadRequest(validationResult.Errors);
            }
            _userService.Create(register);
            return Ok();
        }

        [HttpGet("teachers")]
        [Authorize(Roles = RoleDto.Student)]
        public ActionResult GetTeachers()
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId = int.Parse(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            return Ok(_userService.GetTeachers(userId));
        }

        [HttpPost("students")]
        [Authorize(Roles = RoleDto.Teacher)]
        public ActionResult GetStudents(StudentSearchModel model)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId = int.Parse(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            return Ok(_userService.GetStudents(model.FirstName, model.LastName,
userId));
        }

        [HttpPost("students/add")]
        [Authorize(Roles = RoleDto.Teacher)]
        public ActionResult AddStudent(StudentAddModel model)
        {
            var claimsIdentity = User.Identity as ClaimsIdentity;
            var userId = int.Parse(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);
            _userService.AddStudent(userId, model.StudentId);
            return Ok();
        }
    }
}


using AutoMapper;
using WebApplication.Models;
using WebApplicationBLL.DTOs;
using WebApplicationDAL.Entities;

namespace WebApplication.Helpers
{
    public class AutoMapperProfile : Profile
    {
        public AutoMapperProfile()
        {
            CreateMap<RegisterModel, RegisterDto>();
            CreateMap<LoginModel, LoginDto>();
            CreateMap<RegisterDto, User>();
            CreateMap<User, UserDto>();
            CreateMap<UserDto, User>();
            CreateMap<TaskResultCreateModel, TaskResultCreateDto>();
            CreateMap<TaskCreateModel, TaskCreateDto>();
        }
    }
}


using WebApplicationDAL.Enums;
```

```
namespace WebApplication.Models
{
    public class CodeModel
    {
        public string Command { get; set; }

        public LanguageType LanguageId { get; set; }
    }
}


namespace WebApplication.Models
{
    public class LoginModel
    {
        public string UserName { get; set; }

        public string Password { get; set; }
    }
}


namespace WebApplication.Models
{
    public class RefreshTokenModel
    {
        public string Username { get; set; }
        public string Token { get; set; }
        public bool Revoked { get; set; }
    }
}


namespace WebApplication.Models
{
    public class RegisterModel : LoginModel
    {
             public string FirstName { get; set; }

        public string LastName { get; set; }

        public string Role { get; set; }
    }
}


namespace WebApplication.Models
{
    public class StudentAddModel
    {
        public int StudentId { get; set; }
    }
}


namespace WebApplication.Models
{
    public class StudentSearchModel
    {
        public string FirstName { get; set; }

        public string LastName { get; set; }
    }
}


namespace WebApplication.Models
```

```
{
    public class TaskCreateModel
    {
        public string Description { get; set; }
    }
}
```

```
using WebApplicationDAL.Enums;

namespace WebApplication.Models
{
    public class TaskResultCreateModel
    {
        public string Code { get; set; }

        public LanguageType LanguageId { get; set; }
    }
}
```

```
using WebApplicationDAL.Enums;

namespace WebApplication.Models
{
    public class TaskResultCreateModel
    {
        public string Code { get; set; }

        public LanguageType LanguageId { get; set; }
    }
}
```